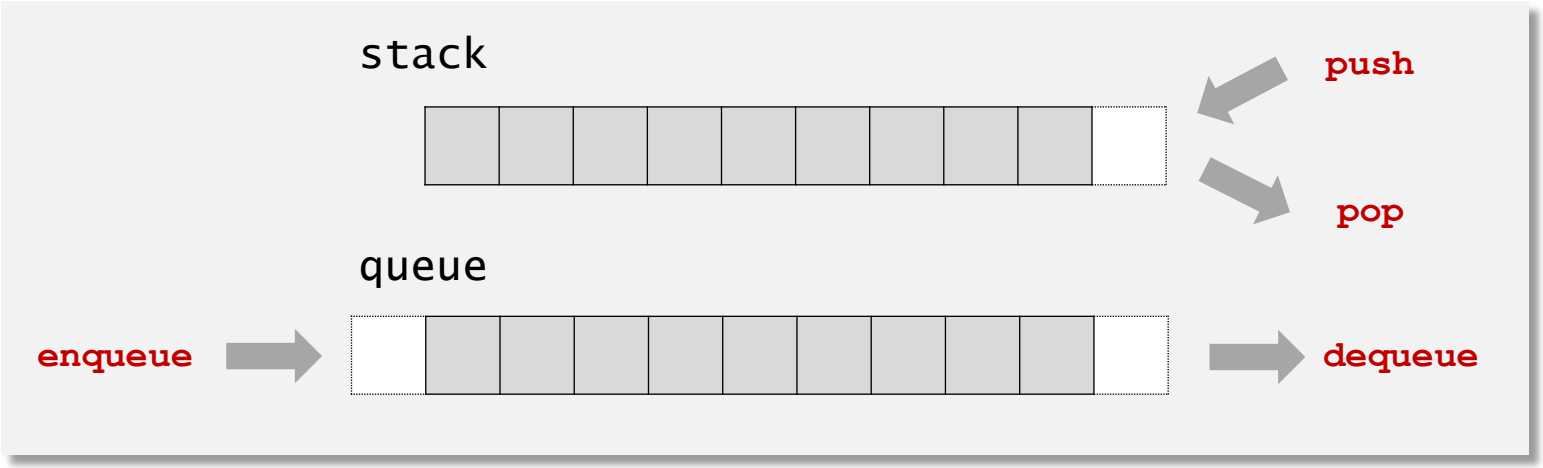
پشته

سید ناصر رضوی www.snrazavi.ir
۱۳۹۵

پشته و صف

- انواع داده‌ای پایه‌ای.
  - محتویات: مجموعه‌ای از اشیا
  - عملیات: درج، حذف، پیمایش، آزمایش خالی بودن
  - منظور از درج واضح است.
  - کدام عنصر باید حذف شود؟



کد کاربر، پیاده‌سازی، واسط

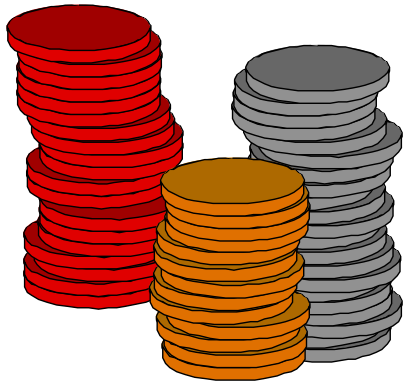
- جداسازی پیاده‌سازی از واسط.
  - مثال: پشته، صف، کیسه، صف اولویت، جدول نماد و ...

- مزایا.
  - کد کاربر از جزییات پیاده‌سازی خبر ندارد، در نتیجه می‌تواند از بین پیاده‌سازی‌های مختلف یکی را انتخاب کند.
  - پیاده‌سازی از نیازهای کد کاربر خبر ندارد، در نتیجه کدهای کاربر زیادی می‌توانند از یک پیاده‌سازی یکسان استفاده کنند .

کد کاربر:	*برنامه‌ای که از عملیات تعریف شده در واسط استفاده می‌کند.*
پیاده‌سازی:	*کد واقعی که عملیات تعریف شده در واسط را پیاده‌سازی می‌کند.*
واسط:	*توصیفی از انواع داده‌ای به همراه عملیات اصلی قابل انجام بر روی آنها.*

پشته

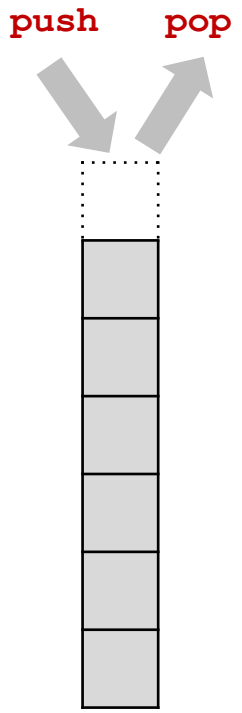
- پشته. لیستی که در آن عمل درج و حذف از یک طرف به نام بالای پشته انجام می‌شود.
  - درج و حذف بر مبنای اصل «آخرین ورودی اولین خروجی»



واسط پشته

- دست‌گرمی. یک پشته از رشته‌ها.

public class StackOfStrings	
StackOfStrings()	ایجاد یک پشته خالی
void push(String s)	درج یک عنصر در بالای پشته
String pop()	حذف عنصر بالای پشته و برگرداندن آن
boolean isEmpty()	بررسی خالی بودن پشته
int size()	برگرداندن تعداد عناصر موجود در پشته



- کد کاربر مثالی. معکوس کردن دنباله‌ی رشته‌های خوانده شده از ورودی.

آزمایش درستی عملکرد پشته

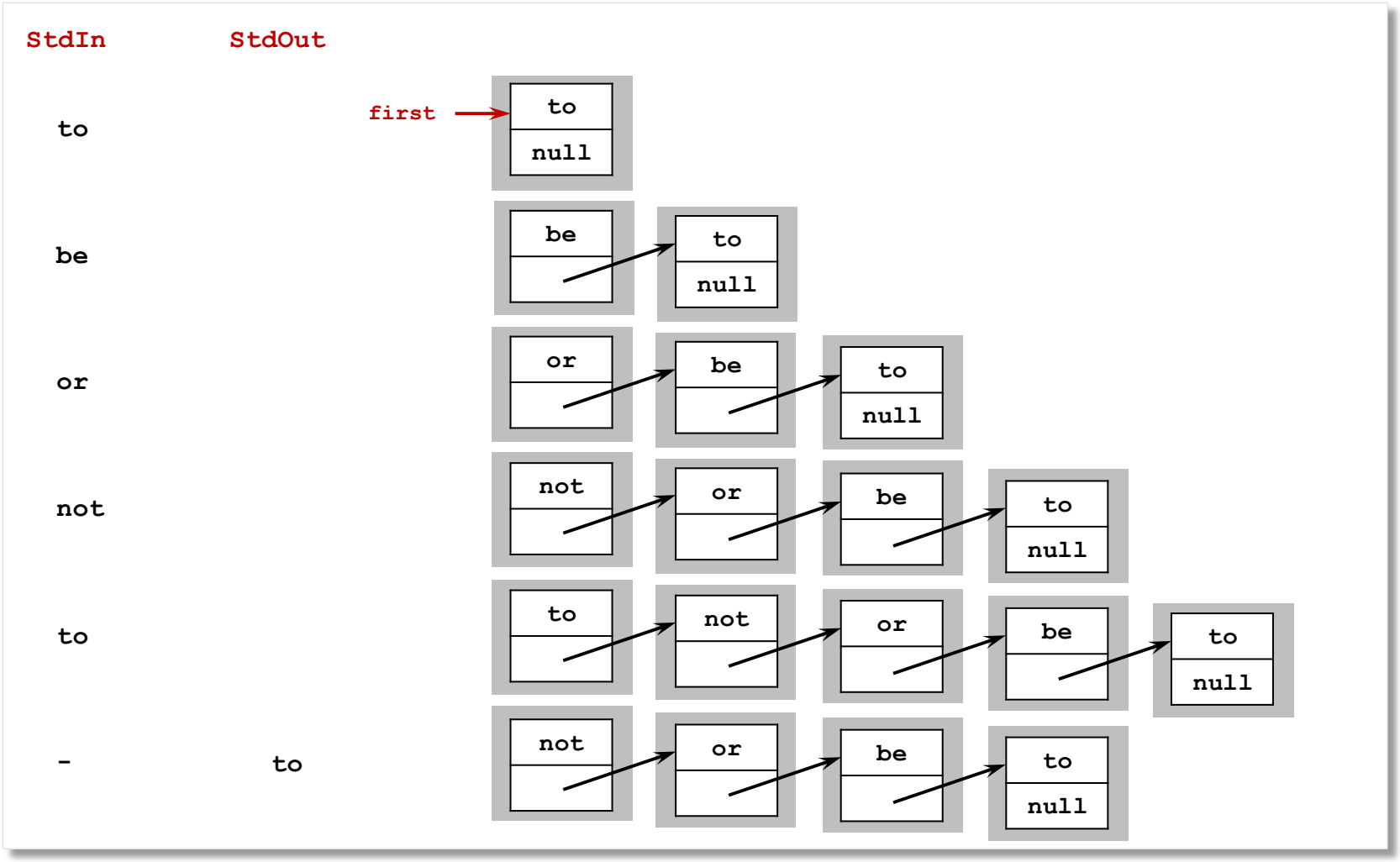
- تعدادی رشته را از ورودی استاندارد بخوان.
  - اگر رشته‌ی ورودی برابر با "-" است، یک رشته از بالای پشته حذف و آن را چاپ کن.
  - در غیر این صورت، رشته‌ی خوانده شده را در بالای پشته درج کن.

```
public static void main(String[] args)
{
   StackOfStrings stack = new StackOfStrings();
   while (!StdIn.isEmpty())
   {
      String item = StdIn.readString();
      if (item.equals("-")) StdOut.print(stack.pop() + " ");
      else                  stack.push(item);
   }
}
```

```
% more tobe.txt
to be or not to - be - - that - - -

% java StackOfStrings < tobe.txt
to be not that or be
```

پشته: پیاده‌سازی با لیست پیوندی

- یک ارجاع به اولین عنصر نگهداری کن؛ درج و حذف را از ابتدای لیست انجام بده.



حذف از پشته

کلاس درونی

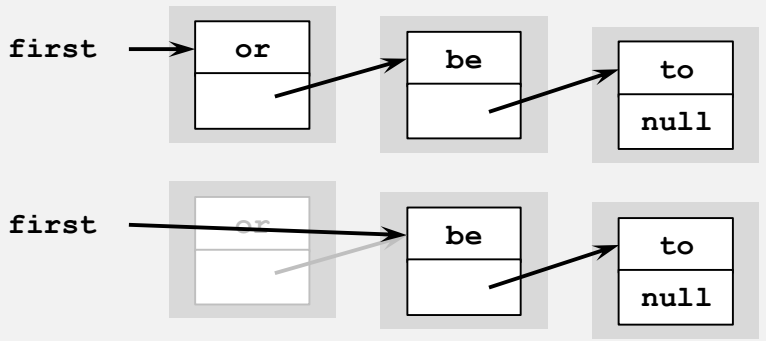
```
private class Node
{
   private String item;
   private Node   next;
}
```

ذخیره‌ی عنصری که باید برگردانده شود

```
String item = first.item;
```

حذف اولین گره

```
first = first.next;
```



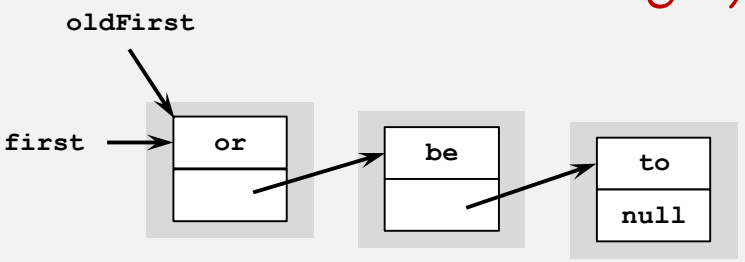
برگرداندن عنصر ذخیره شده

```
return item;
```

درج در پشته

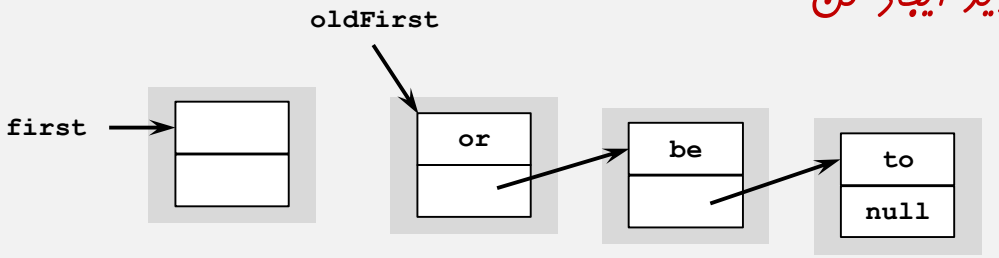
یک ارجاع به اولین عنصر لیست ذخیره کن

```
Node oldFirst = first;
```



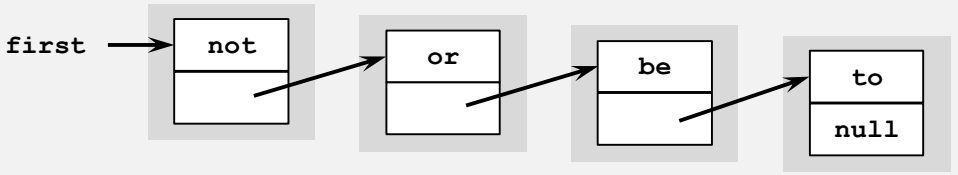
یک گره جدید برای ذخیره‌ی عنصر جدید ایجاد کن

```
first = new node();
```



متغیرهای گره جدید را مقداردهی کن

```
first.item = "not";
first.next = oldFirst;
```



پشته: پیاده‌سازی در جاوا

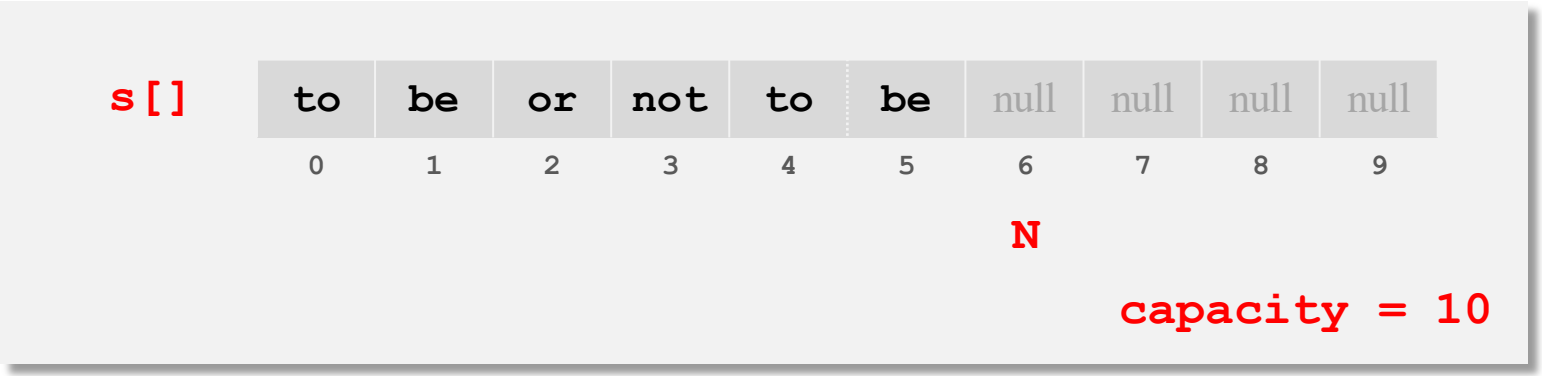

```
public class LinkedStackOfStrings
{
   private Node first = null;

   private class Node {
      private String item;
      private Node next;
   }

   public boolean isEmpty()
   {  return first == null;  }

   public void push(String item) {
      Node oldFirst = first;
      first = new Node();
      first.item = item;
      first.next = oldFirst;
   }

   public String pop() {
      String item = first.item;
      first = first.next;
      return item;
   }
}
```

کلاس درونی

ادعا. تمام عملیات پشته در زمان ثابت انجام می‌شوند.

پشته: پیاده‌سازی به وسیله‌ی آرایه

- پیاده‌سازی پشته به وسیله‌ی آرایه.
  - از آرایه‌ی `s[]` برای ذخیره‌ی `N` عنصر پشته استفاده کن.
  - درج: عنصر جدید را در مکان `s[N]` ذخیره کن.
  - حذف: عنصر واقع در مکان `s[N - 1]` را حذف کن.



- اشکال. اگر تعداد عناصر بیشتر از ظرفیت پشته باشد، سرریز رخ می‌دهد. [در ادامه]

پشته: پیاده‌سازی در جاوا


```
public class FixedCapacityStackOfStrings
{
   private String[] s;
   private int N = 0;

   public FixedCapacityStackOfStrings(int capacity)
   {  s = new String[capacity];  }

   public int size()
   {  return N; }

   public boolean isEmpty()
   {  return N == 0;  }

   public void push(String item)
   {  s[N++] = item;  }

   public String pop()
   {  return s[--N];  }
}
```

تقلب

ملاحظات پشته

- سرریز و پاریز.
  - پاریز: در هنگام حذف از پشته‌ی خالی، یک استثنا بروز دهید.
  - سرریز: اندازه‌ی آرایه را بزرگ‌تر کنید. [در ادامه]
- اتلاف حافظه. نگهداشتن یک ارجاع به شی‌ای که دیگر مورد نیاز نیست.

```
public String pop() {
  if (isEmpty())
     throw new RuntimeException("Empty stack");
  String item = s[--N];
  s[N] = null;
  return item;
}
```

- اشیای پوچ. در پیاده‌سازی فعلی درج یک شیء پوچ مجاز است.

پشته: تغییر اندازه‌ی آرایه

- مشکل. تنظیم اندازه‌ی پشته به وسیله‌ی کد کاربر.
- س. اندازه‌ی پشته‌ی جدید چقدر باشد؟
  - استراتژی خسیس. اندازه‌ی آرایه به اندازه‌ی یک مقدار ثابت افزایش یابد:

```
f(N) = N + c
```

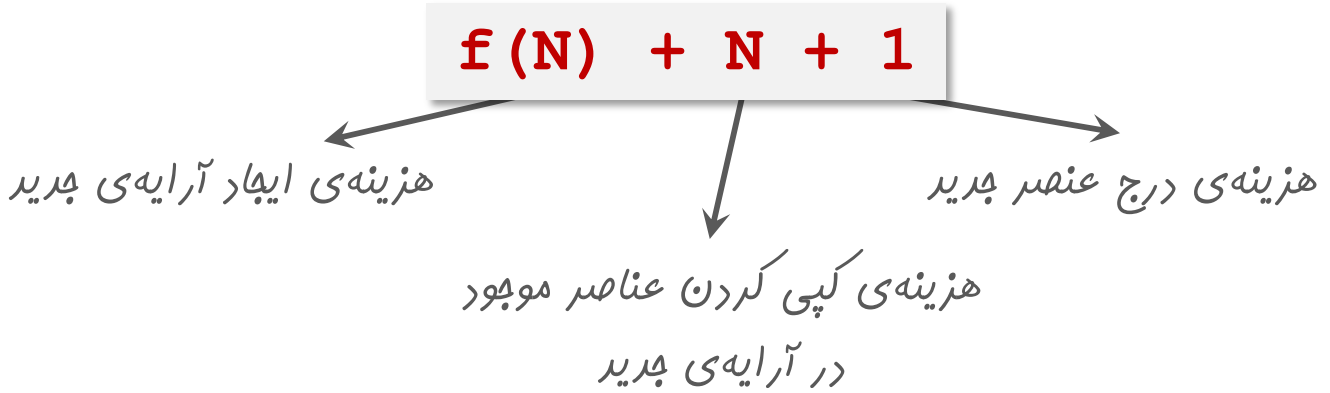
  - استراتژی رشد. اندازه‌ی آرایه دو برابر شود:

```
f(N) = N * 2
```

- س. کدام استراتژی بهتر است؟

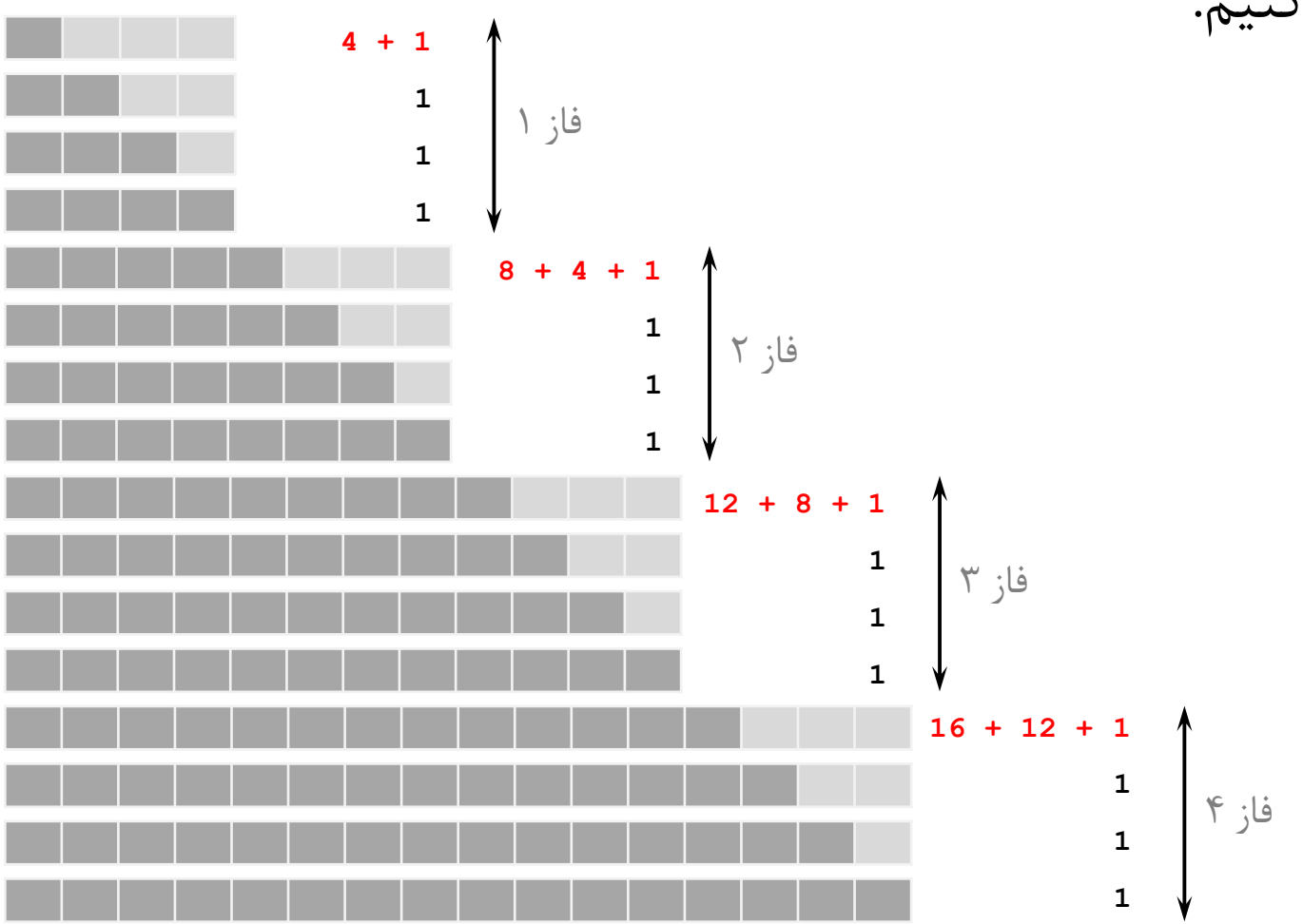
مقایسه استراتژی های خسیس و رشد

- به منظور مقایسه‌ی این دو استراتژی، از مدل هزینه‌ی زیر استفاده می‌کنیم:
  - عمل درج معمولی: یک عنصر جدید به بالای پشته اضافه می‌کند و دارای هزینه‌ای به اندازه‌ی یک واحد است.
  - عمل درج ویژه: یک آرایه‌ی جدید با اندازه‌ی **f(N)** ایجاد می‌کند، **N** عنصر موجود را در آرایه‌ی جدید کپی می‌کند و سپس عنصر جدید را در بالای پشته درج می‌کند. هزینه‌ی این عملیات برابر است با:



استراتژی خسیس (c = 4)

- با یک آرایه با اندازه‌ی صفر شروع می‌کنیم.



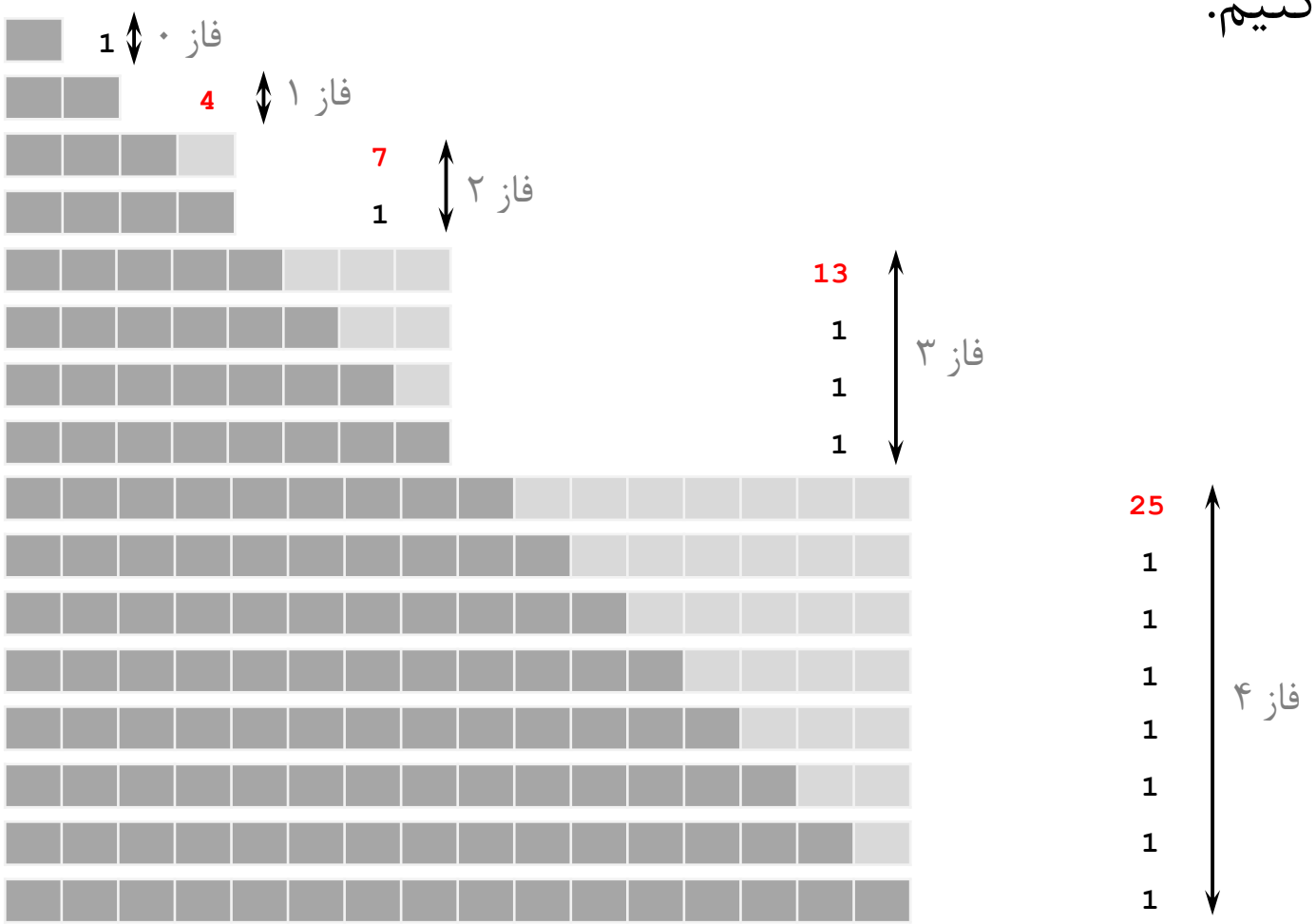
کارایی استراتژی خسیس

- در فاز i اندازه‌ی آرایه برابر است با $c \times i$.
- هزینه‌ی کل فاز i برابر است با:
  - هزینه‌ی ایجاد آرایه جدید: $c \times i$
  - هزینه‌ی کپی کردن عناصر: $c \times (i-1)$
  - هزینه‌ی درج عناصر جدید: c
- در نتیجه، هزینه‌ی کل فاز i برابر است با $2ci$.
- از آنجا که تعداد کل فازها برای درج N عنصر برابر است با N/c، در نتیجه هزینه‌ی کل فازها برابر است با:

$$2c(1+2+\cdots+N/c) \in O(N^2/c)$$

استراتژی رشد

- با یک آرایه با اندازه‌ی یک شروع می‌کنیم.



کارایی استراتژی رشد

- در فاز i اندازه‌ی آرایه برابر است با 2^i .
- هزینه‌ی کل فاز i برابر است با:
  - هزینه‌ی ایجاد آرایه جدید: 2^i
  - هزینه‌ی کپی کردن عناصر: 2^{i-1}
  - هزینه‌ی درج عناصر جدید: 2^{i-1}
- در نتیجه هزینه‌ی کل فاز i برابر است با 2^{i+1}.
- از آنجا که تعداد کل فازها برای درج N عنصر برابر است با $\lg N + 1$، در نتیجه هزینه‌ی کل فازها برابر است با:

$$2^1 + 2^2 + 2^3 + \cdots + 2^{\lg N+1} = 2^{\lg N+2} - 2 = 4N - 2 \in O(N)$$

تحلیل سرشکن شده

- تحلیل سرشکن شده. محاسبه‌ی هزینه دنباله‌ای از N عمل.
- هدف. نشان دادن این که اگرچه در دنباله عملیات انجام شده برخی از عمل‌ها دارای هزینه‌ی نسبتاً بالایی هستند، اما در کل هر عمل به طور سرشکن شده هزینه‌ی پایینی دارد.
- روشهای تحلیل سرشکن شده.
  - روش جمع زنی
  - روش حسابداری
  - روش پتانسیل

تحلیل سرشکن شده: مثال

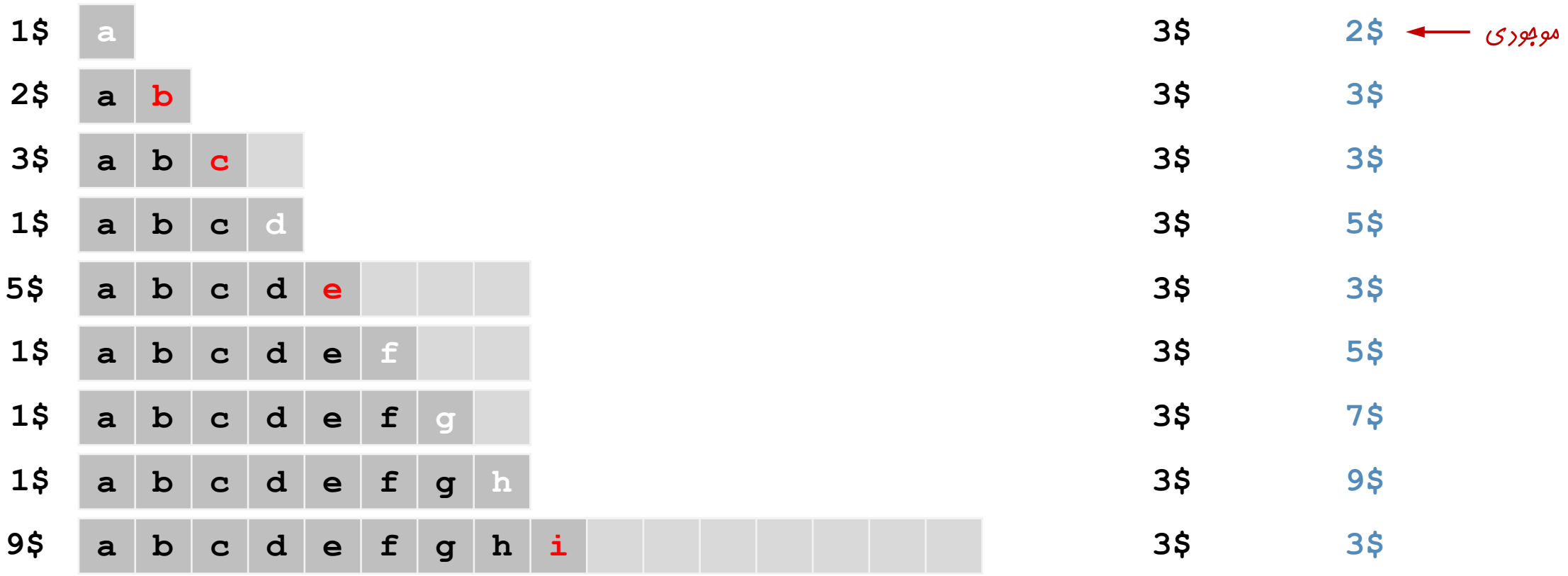
- محاسبه‌ی هزینه‌ی سرشکن شده‌ی هر عمل درج در پشته‌های قابل رشد:

  - روش جمع زنی.
    - مجموع هزینه‌ی **N** عمل درج همان گونه که دیدید برابر است با **4N - 2** .
    - در نتیجه، هزینه‌ی سرشکن شده‌ی هر عمل درج برابر است با **O(1)** .

  - روش حسابداری.
    - در این روش برای انجام برخی از عملیات در دنباله، هزینه‌ی بیشتری از آنچه که واقعاً نیاز است می‌پردازیم. هزینه‌های اضافی پرداخت شده در حساب ما ذخیره می‌شوند. سپس در هنگام انجام یک عمل پر هزینه می‌توانیم مقداری از هزینه را از موجودی حساب خود بپردازیم. [به شرطی که موجودی مثبت باشد]

تحلیل سرشکن شده به روش حسابداری: مثال

- برای هر عمل درج ۳ دلار می‌پردازیم. [بدون در نظر گرفتن هزینه‌ی ایجاد آرایه]



پشته‌ی قابل رشد: پیاده‌سازی

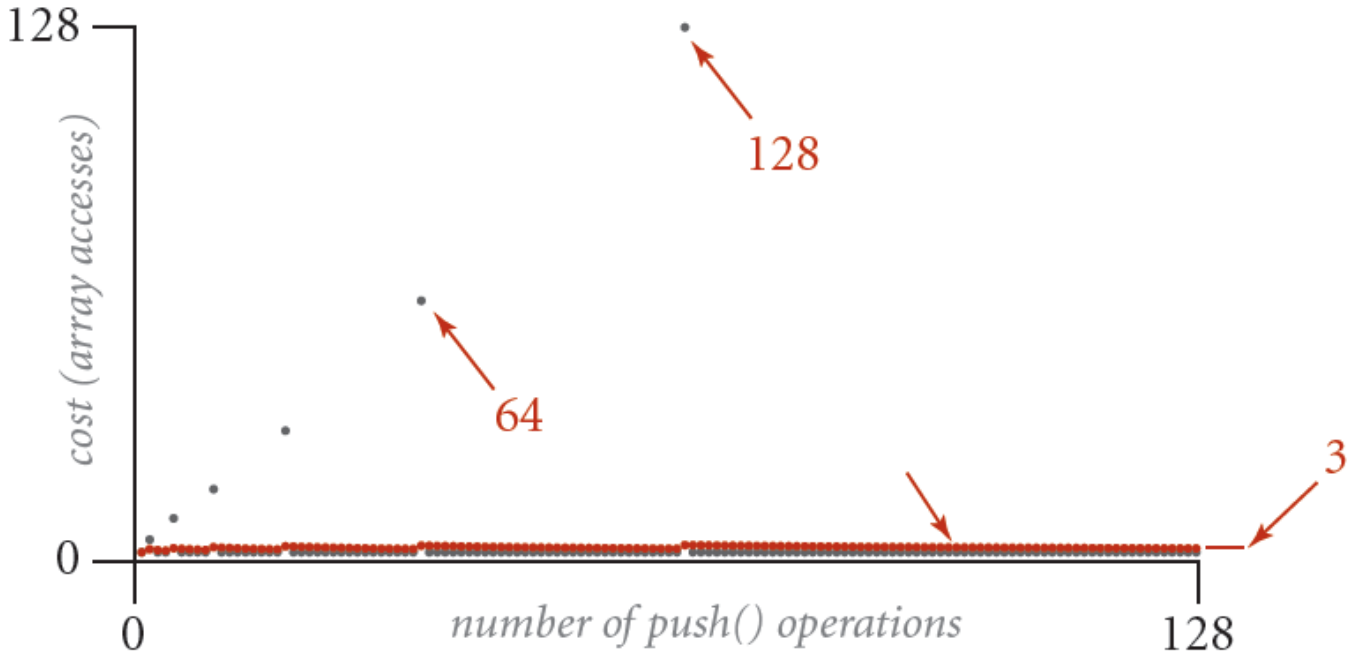

```
public class ResizingArrayStackOfStrings
{
   private String[] s;
   private int N = 0;

   public ResizingArrayStackOfStrings()
   {  s = new String[1];  }

   public void push(String item)
   {
      if (N == s.length) resize(2 * s.length);
      s[N++] = item;
   }

   public void resize(int capacity)
   {
      String[] copy = new String[capacity];
      for (int i = 0; i < N; i++)
         copy[i] = s[i];
      s = copy;
   }
   ...
}
```

پشته: هزینه سرشکن شده درج

- هزینه‌ی درج **N** عنصر اول. [بدون در نظر گرفتن هزینه‌ی ایجاد آرایه جدید]

$$N + (1 + 2 + 4 + \cdots + N) \sim 3N$$

هزینه‌ی درج‌ها هزینه‌ی کپی‌ها



پشته‌ی قابل رشد: کوچک کردن آرایه

- س. چگونه اندازه‌ی آرایه را کوچک کنیم؟

- راه‌حل اول.
  - درج: وقتی آرایه پر است، اندازه‌ی آن را دو برابر می‌کنیم.
  - حذف: وقتی آرایه نیمه پر است، اندازه‌ی آن را نصف می‌کنیم.

- این روش در بدترین حالت بسیار پر هزینه است!
  - دنباله‌ای از عملیات درج، حذف، درج، حذف، درج، حذف و ... را وقتی که آرایه پر است در نظر بگیرید؛
  - در این حالت هر عمل دارای هزینه‌ای متناسب با **N** است!

پشته‌ی قابل رشد: کوچک کردن آرایه

- س. چگونه اندازه‌ی آرایه را کوچک کنیم؟

- راه‌حل کارا.
  - درج: وقتی آرایه پر است، اندازه‌ی آن را دو برابر می‌کنیم.
  - حذف: وقتی یک چهارم آرایه پر است، اندازه‌ی آن را نصف می‌کنیم.

```
public String pop()
{
   String item = s[--N];
   s[N] = null;
   if (N > 0 && s.length = 4 * N) resize( s.length / 2 );
   return item;
}
```

پشته با قابلیت تغییر اندازه: مثال

push()	pop()	N	s.length	s[]							
				0	1	2	3	4	5	6	7
		0	1	null							
to		1	1	to							
be		2	2	to	be						
or		3	4	to	be	or	null				
not		4	4	to	be	or	not				
to		5	8	to	be	or	not	to	null	null	null
-	to	4	8	to	be	or	not	null	null	null	null
be		5	8	to	be	or	not	be	null	null	null
-	be	4	8	to	be	or	not	null	null	null	null
-	not	3	8	to	be	or	null	null	null	null	null
that		4	8	to	be	or	that	null	null	null	null
-	that	3	8	to	be	or	null	null	null	null	null
-	or	2	4	to	be	null	null				
-	be	1	2	to	null						
is		2	2	to	is						

پیاده‌سازی پشته: لیست پیوندی یا آرایه؟

- س. کدام پیاده‌سازی بهتر است؟

- پیاده‌سازی با لیست پیوندی.
  - هزینه‌ی هر عمل در بدترین حالت ثابت
  - دارای سربار زمانی و حافظه به دلیل وجود پیوندها

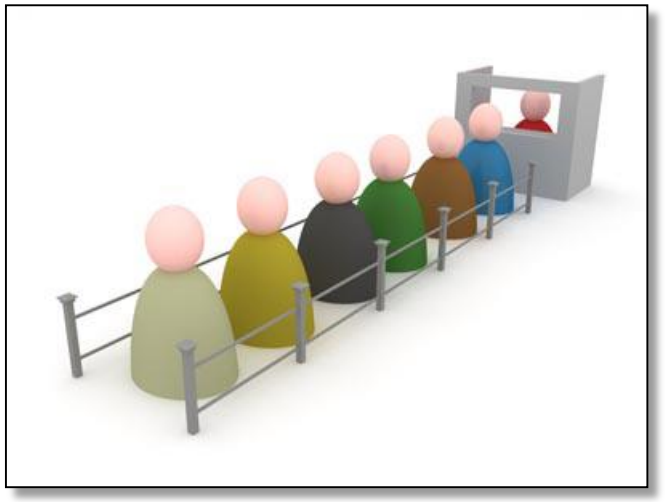
- پیاده‌سازی با آرایه با قابلیت تغییر اندازه.
  - هزینه‌ی سرشکن شده‌ی هر عمل ثابت
  - اتلاف حافظه‌ی کمتر

صف

صف

- تفاوت صف با پشته در این است که درج و حذف عناصر در صف از اصل «اولین ورودی، اولین خروجی» پیروی می‌کند.

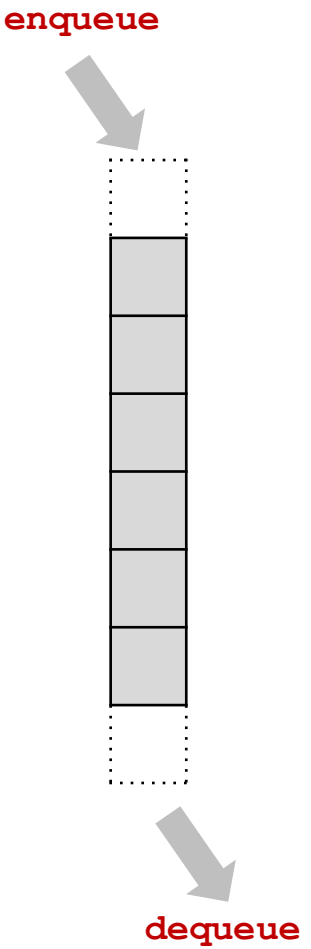
- به عبارت دیگر هر زمان که بخواهیم می‌توانیم یک عنصر به انتهای صف اضافه کنیم، اما در هنگام حذف تنها اجازه داریم اولین عنصر (عنصری که بیشتر از بقیه‌ی عناصر در صف بوده است) را از صف حذف کنیم.



- عملیات اصلی صف.
  - درج یک عنصر در انتهای صف
  - حذف یک عنصر از ابتدای صف

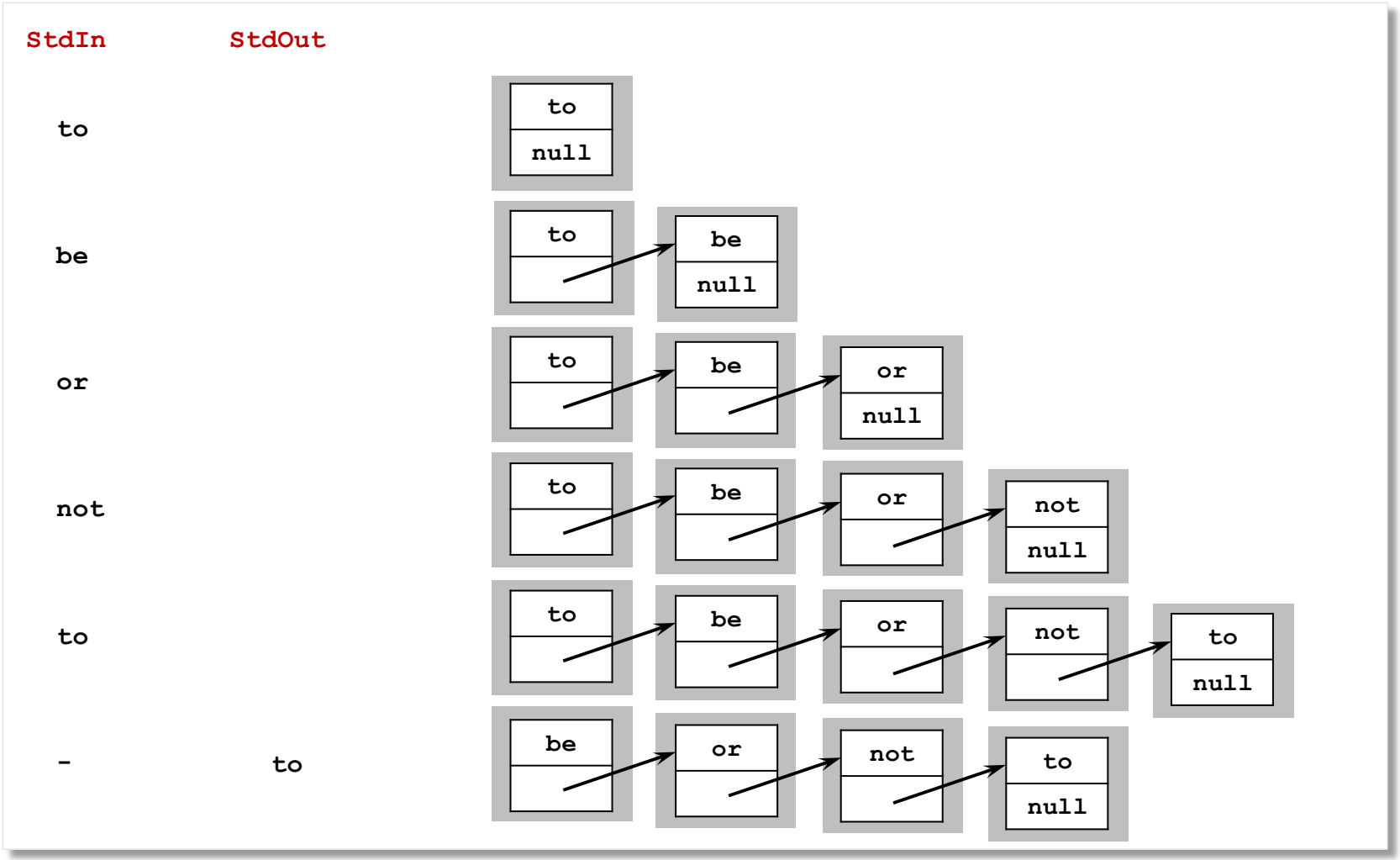
واسط صف

public class QueueOfStrings	
QueueOfStrings()	ایجاد یک صف خالی
void enqueue(String s)	درج یک عنصر در انتهای صف
String dequeue()	حذف یک عنصر از ابتدا و برگرداندن آن
boolean isEmpty()	بررسی خالی بودن صف
int size()	برگرداندن تعداد عناصر موجود در صف



صف: پیاده‌سازی با لیست پیوندی

- یک ارجاع به اولین عنصر و آخرین عنصر در لیست پیوندی نگهداری کن.



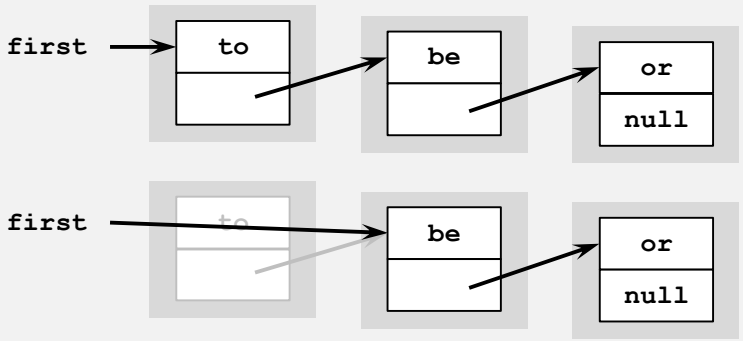
حذف از صف

ذخیره‌ی عنصری که باید برگردانده شود

```
String item = first.item;
```

حذف اولین گره

```
first = first.next;
```



برگرداندن عنصر ذخیره شده

```
return item;
```

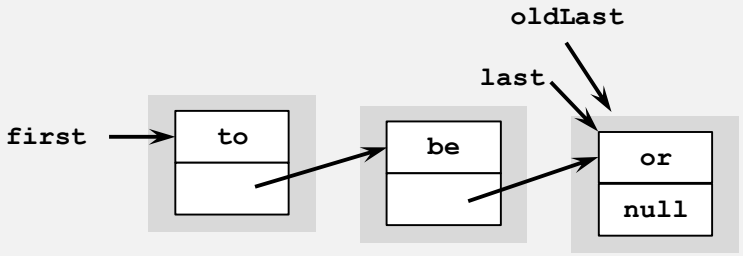
کلاس درونی

```
private class Node
{
   private String item;
   private Node   next;
}
```

درج در صف

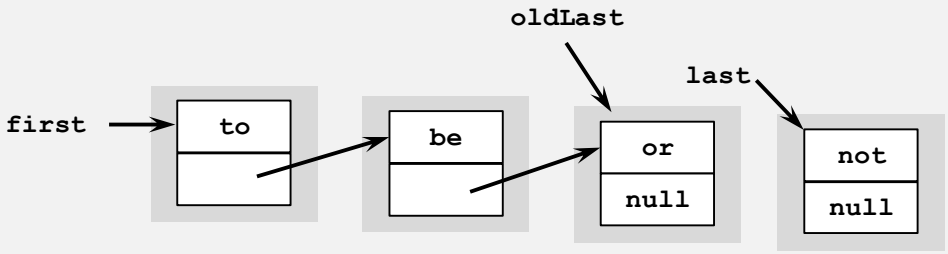
یک ارجاع به آخرین عنصر لیست ذخیره کن

```
Node oldLast = last;
```



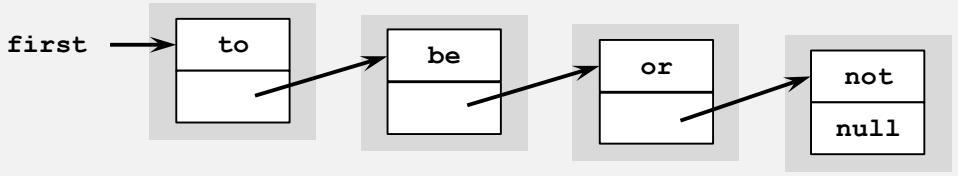
یک گره جدید برای انتهای صف ایجاد کن

```
last = new node();
last.item = "not";
```



گره جدید را به انتهای صف وصل کن

```
oldLast.next = last;
```



صف: پیاده‌سازی در جاوا


```
public class LinkedQueueOfStrings {
   private Node first, last;

   private class Node
   {  /* same as before */  }

   public boolean isEmpty()
   {  return first == null;  }

   public void enqueue(String item) {
      Node oldLast = last;
      last = new Node();
      last.item = item;
      if (isEmpty()) first = last;
      else           oldLast.next = last;
   }

   public String dequeue() {
      String item = first.item;
      first       = first.next;
      if (isEmpty()) last = null;
      return item;
   }
}
```

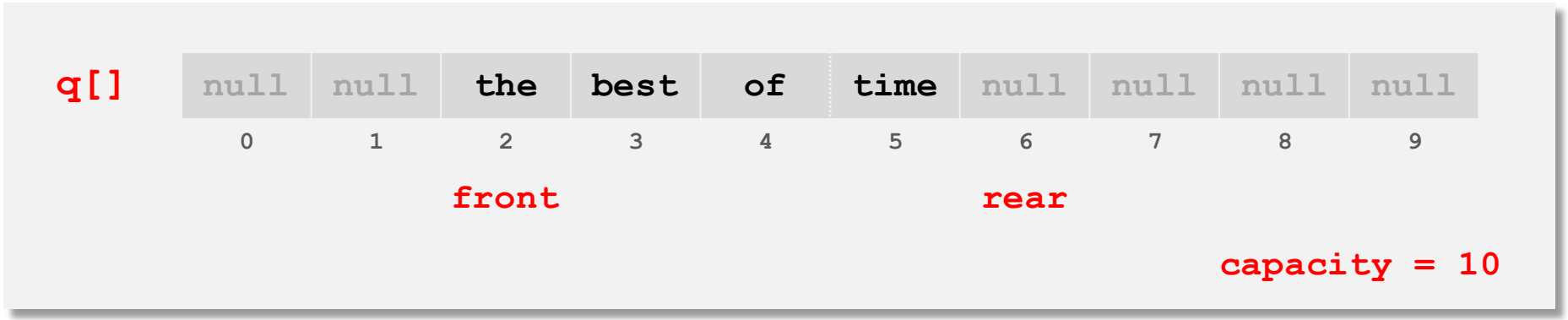
کلاس درونی

ادعا. تمام عملیات صف در زمان ثابت انجام می‌شوند.

صف: پیاده‌سازی با آرایه

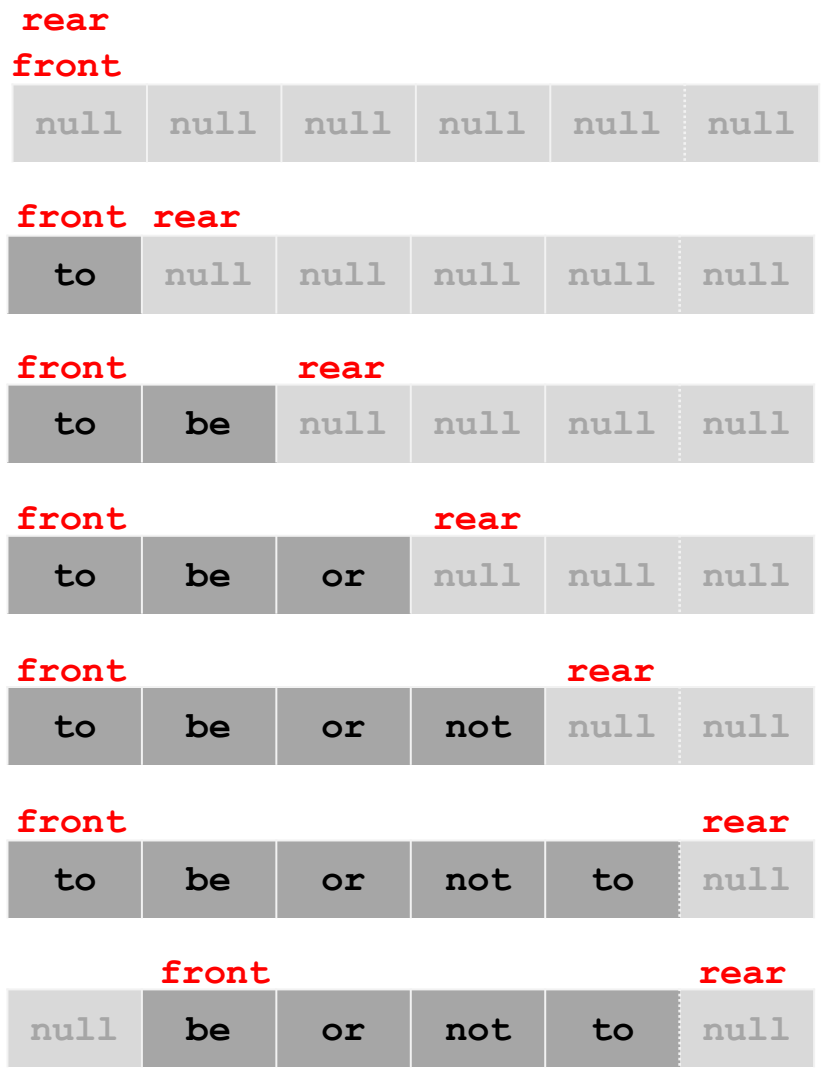
- پیاده‌سازی صف به وسیله‌ی آرایه.
  - از آرایه‌ی **q[]** برای ذخیره‌ی عناصر صف استفاده کن.
  - درج: عنصر جدید را در مکان **q[rear]** ذخیره کن.
  - حذف: عنصر واقع در مکان **q[front]** را حذف کن.
  - اندیس‌های **front** و **rear** را به پیمانه‌ی ظرفیت افزایش بده.
  - به آرایه قابلیت تغییر اندازه را اضافه کن.

```
front = (front + 1) % capacity
```



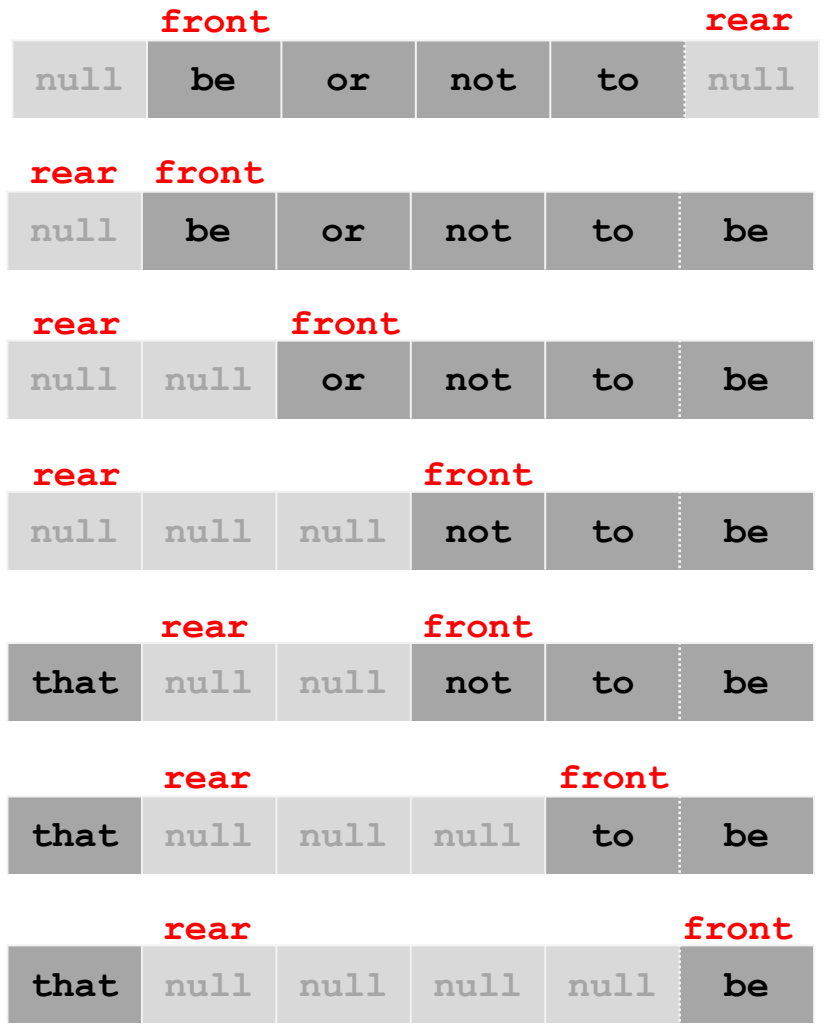
صف: مثال

StdIn	StdOut
to	
be	
or	
not	
to	
-	to



صف: مثال

StdIn	StdOut
be	
-	be
-	or
that	
-	not
-	to



تمرین: پیاده‌سازی صف با آرایه

- پیاده‌سازی صف با آرایه
  - با قابلیت کوچک و بزرگ شدن آرایه

تمرین: پیاده‌سازی صف با آرایه


```
public class ResizingArrayQueueOfStrings
{
   private String[] q;
   private int front;
   private int rear;

   public ResizingArrayQueueOfStrings()      {  ...  }

   public int size()                         {  ...  }

   public boolean isEmpty()                  {  ...  }

   public void enqueue(String item)          {  ...  }

   public String dequeue()                   {  ...  }

   public void resize(int capacity)          {  ...  }

   private int next(int index)
   {
      return (index + 1) % q.length;
   }

}
```

انواع عمومی (generics)

پشته پارامتری

- تا اینجا یک پشته به منظور ذخیره‌ی رشته‌ها پیاده‌سازی کردیم.
- س. اما اگر پشته‌ای از اعداد صحیح یا پشته‌ای از **URL**ها بخواهیم، چه می‌شود؟

- تلاش اول. برای هر نوع داده‌ای، یک پشته‌ی مجزا تعریف کن؛
  - بازنویسی کد خسته کننده است و می‌تواند منجر به بروز خطا گردد.
  - کپی کردن کد نیز خسته کننده است و می‌تواند منجر به بروز خطا گردد.

- این روش تا قبل از **java 1.5** معقول‌ترین روش بود!

پشته پارامتری

- تا اینجا یک پشته به منظور ذخیره‌ی رشته‌ها پیاده‌سازی کردیم.
- س. اما اگر پشته‌ای از اعداد صحیح یا پشته‌ای از **URL**ها بخواهیم، چه می‌شود؟

- تلاش دوم . عناصر پشته را از نوع **Object** تعریف کن.
  - در کد کاربر نیاز به تبدیل نوع داریم.
  - تبدیل نوع می‌تواند باعث بروز خطا در زمان اجرا گردد. [عدم همخوانی نوع]

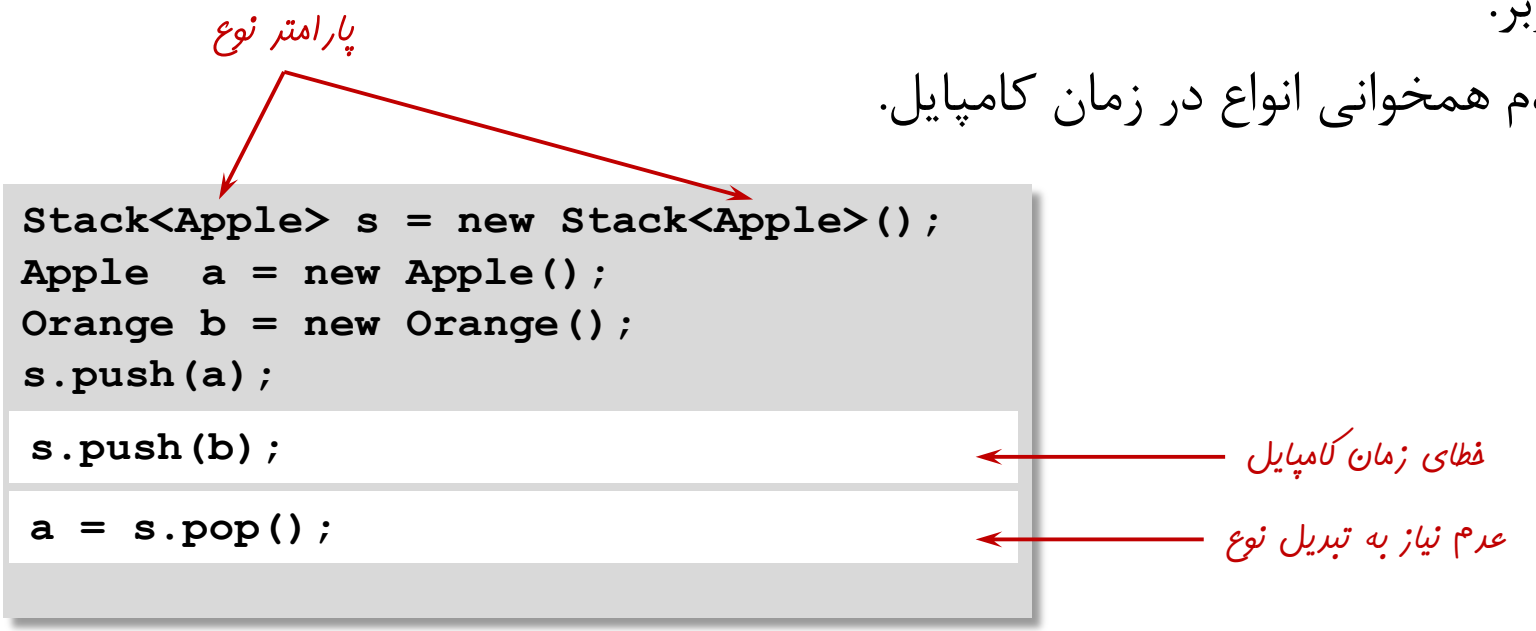
```
StackOfObjects s = new
StackOfObjects();
Apple  a = new Apple();
Orange b = new Orange();
s.push(a);
s.push(b);
a = (Apple) s.pop();
```

خطای زمان اجرا

پشته پارامتری

- تا اینجا یک پشته به منظور ذخیره‌ی رشته‌ها پیاده‌سازی کردیم.
- س. اما اگر پشته‌ای از اعداد صحیح یا پشته‌ای از **URL** ها بخواهیم، چه می‌شود؟

- تلاش سوم . استفاده از انواع عمومی جاوا.
  - اجتناب از تبدیل نوع در کد کاربر.
  - تشخیص خطاهای مربوط به عدم همخوانی انواع در زمان کامپایل.



پشته‌ی پارامتری: پیاده‌سازی با لیست پیوندی


```
public class LinkedStackOfStrings {
   private Node first = null;

  private class Node {
     private String item;
     private Node next;
  }

   public boolean isEmpty()
   {  return first = null;  }

   public void push(String item) {
      Node oldFirst = first;
      first = new Node();
      first.item = item;
      first.next = oldFirst;
   }

   public String pop() {
      String item = first.item;
      first = first.next;
      return item;
   }
}
```

```
public class Stack<Item> {
   private Node first = null;

  private class Node {
     private Item item;
     private Node next;
  }

   public boolean isEmpty()
   {  return first = null;  }

   public void push(Item item) {
      Node oldFirst = first;
      first = new Node();
      first.item = item;
      first.next = oldFirst;
   }

   public Item pop() {
      Item item = first.item;
      first = first.next;
      return item;
   }
}
```

پشته‌ی پارامتری: پیاده‌سازی با آرایه


```
public class FixedCapacityStackOfStrings
{
   private String[] s;
   private int N = 0;

   public ..StackOfStrings(int capacity)
   {  s = new String[capacity];  }

   public int size()
   {  return N; }

   public boolean isEmpty()
   {  return N == 0;  }

   public void push(String item)
   {  s[N++] = item;  }

   public String pop()
   {  return s[--N];  }
}
```

```
public class FixedCapacityStack<Item>
{
   private Item[] s;
   private int N = 0;

   public FixedCapacityStack(int capacity)
   {  s = new Item[capacity];  }

   public int size()
   {  return N; }

   public boolean isEmpty()
   {  return N == 0;  }

   public void push(Item item)
   {  s[N++] = item;  }

   public Item pop()
   {  return s[--N];  }
}
```

شکل نادرست

در جاوا ایجاد یک آرایه از انواع عمومی مجاز نیست!!!

پشته‌ی پارامتری: پیاده‌سازی با آرایه


```
public class FixedCapacityStackOfStrings
{
   private String[] s;
   private int N = 0;

   public ..StackOfStrings(int capacity)
   {  s = new String[capacity];  }

   public int size()
   {  return N; }

   public boolean isEmpty()
   {  return N == 0;  }

   public void push(String item)
   {  s[N++] = item;  }

   public String pop()
   {  return s[--N];  }
}
```

```
public class FixedCapacityStack<Item>
{
   private Item[] s;
   private int N = 0;

   public FixedCapacityStack(int capacity)
   {  s = (Item[]) new Object[capacity];  }

   public int size()
   {  return N; }

   public boolean isEmpty()
   {  return N == 0;  }

   public void push(Item item)
   {  s[N++] = item;  }

   public Item pop()
   {  return s[--N];  }
}
```

شکل درست

تبدیل یک آرایه از **Object** ها به یک آرایه از **Item** ها

انواع داده‌ای عمومی: تبدیل خودکار به شی

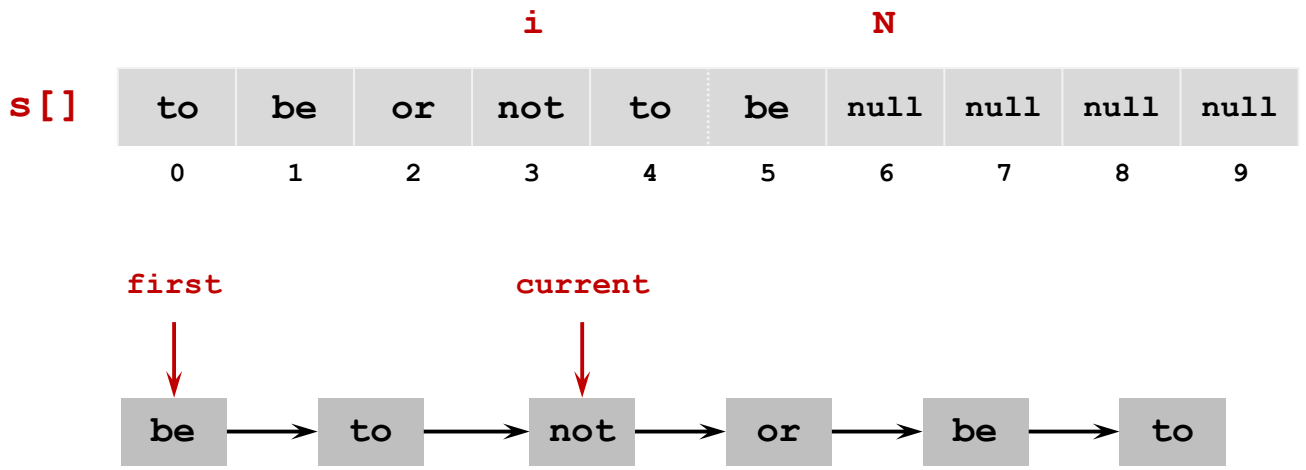
- س. آیا می‌توان در پشته‌ی پارامتری انواع داده‌ای اولیه را ذخیره نمود؟
- انواع بسته بندی کننده **(Wrapper)**
  - هر نوع داده‌ای اولیه دارای یک بسته بندی کننده از نوع شی است.
  - مثال: شی **Integer** بسته بندی کننده نوع داده‌ای اولیه‌ی **int** است.
- بسته بندی خودکار. تبدیل خودکار یک نوع داده‌ای اولیه به نوع بسته بندی کننده.

```
Stack<Integer> s = new Stack<Integer>();
s.push(17);              // s.push(new Integer(17));
int a = s.pop();         // int a = s.pop().intValue();
```

تکرارگرها (iterators)

تکراگرها

- چالش طراحی. اجازه دادن به کد کاربر برای حلقه زدن بر روی عناصر پشته بدون آشکار کردن بازنمایی داخلی پشته و جزییات پیاده‌سازی آن.



- راه‌حل جاوا. پیاده‌سازی واسط **Iterable**

تکرارگرها

- س. یک شیء **Iterable** چیست؟
- ج. هر شی با یک متد که یک **Iterator** برگرداند.

```
public Interface Iterable<Item>
{
   Iterator<Item> iterator();
}
```

- س. **Iterator** چیست؟
- ج. دارای متدهای **hasNext()** و **next()** است.

```
public Interface Iterator<Item>
{
   boolean hasNext();
   boolean next();
   void remove();
}
```

- س. چرا یک ساختمان داده باید **Iterable** باشد؟
- ج. نوشتن حلقه‌های ساده و زیبا

```
Iterator<String> it = stack.iterator();
while (it.hasNext())
{
   String s = it.next();
   StdOut.println(s);
}
```

```
for (String s : stack)
   StdOut.println(s);
```

تکرارگر پشته: پیاده‌سازی لیست پیوندی

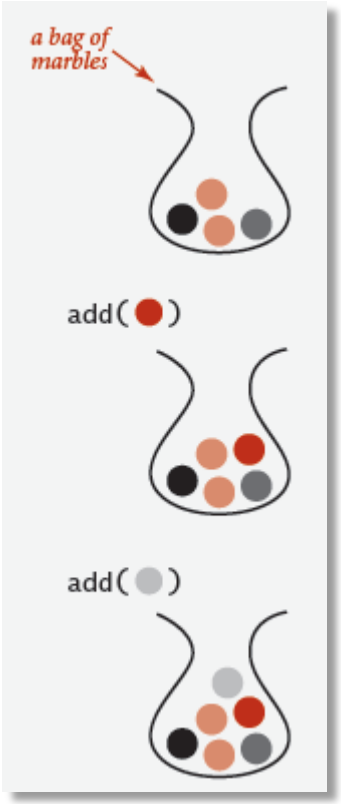

```
import java.util.Iterator;

public class Stack<Item> implements Iterable<Item>
{
   ...
   public Iterator<Item> iterator()
   { return new ListIterator(); }

   private class ListIterator implements Iterator<Item>
   {
      private Node current = first;

      public boolean hasNext() {  return current != null;  }
      public void remove()     {  /* not supported */      }
      public Item next()
      {
         Item item = current.item;
         current = current.next;
         return item;
      }
   }
}
```

تکرارگر پشته: پیاده‌سازی آرایه


```
import java.util.Iterator;

public class Stack<Item> implements Iterable<Item>
{
   ...
   public Iterator<Item> iterator()
   { return new ReverseArrayIterator(); }

   private class ReverseArrayIterator implements Iterator<Item>
   {
      private int i = N;

      public boolean hasNext() {  return i > 0;        }
      public void remove()     {  /* not supported */ }
      public Item next()       {  return s[--i];       }
   }
}
```

واسط کیسه

- کاربرد اصلی. افزودن عناصر به یک کلکسیون و تکرار روی عناصر. [زمانی که ترتیب مهم نباشد]



public class Bag<Item> implements Iterable<Item>	
Bag()	ایجاد یک کیسه خالی
void add(Item x)	درج یک عنصر کیسه
int size()	برگرداندن تعداد عناصر موجود در کیسه
Iterable<Item> iterator()	تکرارگر بر روی تمام عناصر موجود

- پیاده‌سازی. پشته (بدون **pop**) یا صف (بدون **dequeue**)

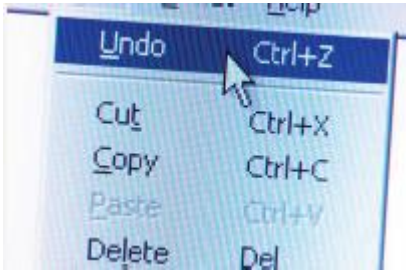
کاربردها

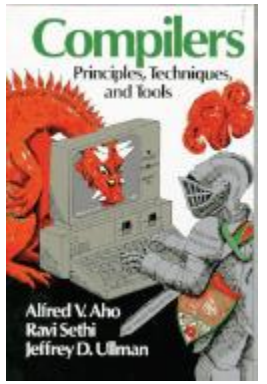
کاربردهای پشته

کاربردهای صف

کاربردهای پشته

- تجزیه در کامپایلر
- ماشین مجازی جاوا
- عملگر **undo** در یک واژه‌پرداز
- دکمه‌ی برگشت در یک مرورگر وب
- پیاده‌سازی فراخوانی توابع در یک کامپایلر
- ...





فراخوانی توابع

- چگونه کامپایلر اجرای توابع را ممکن می‌سازد؟
  - فراخوانی تابع: ذخیره مقادیر متغیرهای محلی و آدرس بازگشت در پشته
  - برگشت: برداشتن آدرس بازگشت و مقادیر متغیرهای محلی از پشته

- تابع بازگشتی. تابعی که خودش را فراخوانی می‌کند.
  - همواره می‌توان از پشته برای حذف فراخوانی‌های بازگشتی استفاده نمود.

gcd(216, 192)

```
public static int gcd(int p, int q) {
   if (q == 0) return p;
   else
}
```

gcd(192, 24)

```
public static int gcd(int p, int q) {
   if (q == 0) return p;
   else
}
```

gcd(24, 0)

```
public static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```

ارزیابی عبارت‌های ریاضی

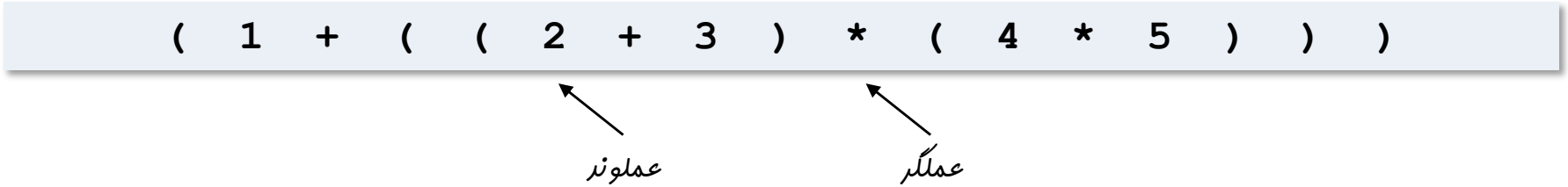
- هدف. ارزیابی عبارت‌های میانوندی.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```

- الگوریتم دو پشته‌ای. [دیکسترا]
  - عملوند: درج در پشته‌ی مقدار؛
  - عملگر: درج در پشته‌ی عملگر؛
  - پرانتز باز: نادیده بگیر؛
  - پرانتز بسته: یک عملگر و دو عملوند از روی پشته‌ها حذف کن. عملگر را بر روی عملوندها اعمال کن و حاصل را دوباره در پشته‌ی مقدار درج کن.

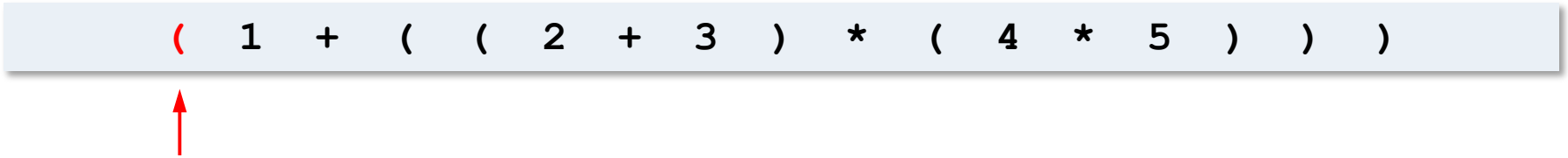
ارزیابی عبارت‌های میانوندی

- الگوریتم دو پشته‌ای. [دیکسترا]
  - عملوند: درج در پشته‌ی مقدار؛
  - عملگر: درج در پشته‌ی عملگر؛
  - پرانتز باز: نادیده بگیر؛
  - پرانتز بسته: یک عملگر و دو عملوند از روی پشته‌ها حذف کن. عملگر را بر روی عملوندها اعمال کن و حاصل را دوباره در پشته‌ی مقدار درج کن.



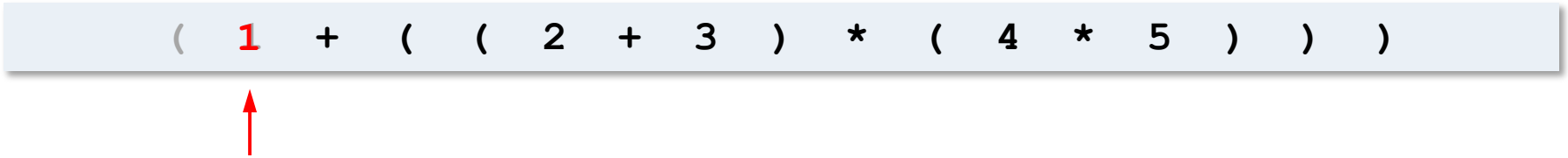
ارزیابی عبارت‌های میانوندی

پشته عملگر پشته مقدار



ارزیابی عبارت‌های میانوندی

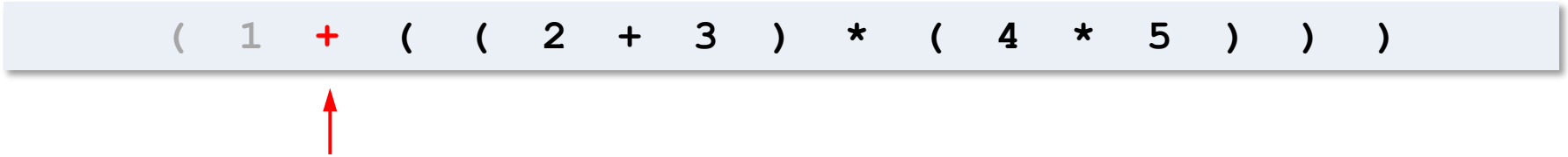
پشته عملگر پشته مقدار



ارزیابی عبارت‌های میانوندی

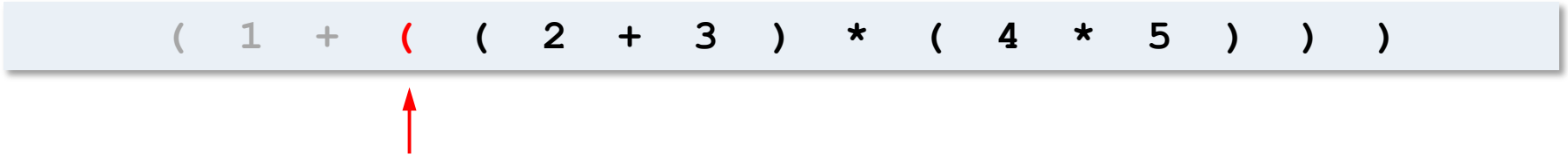
1

پشته عملگر پشته مقدار

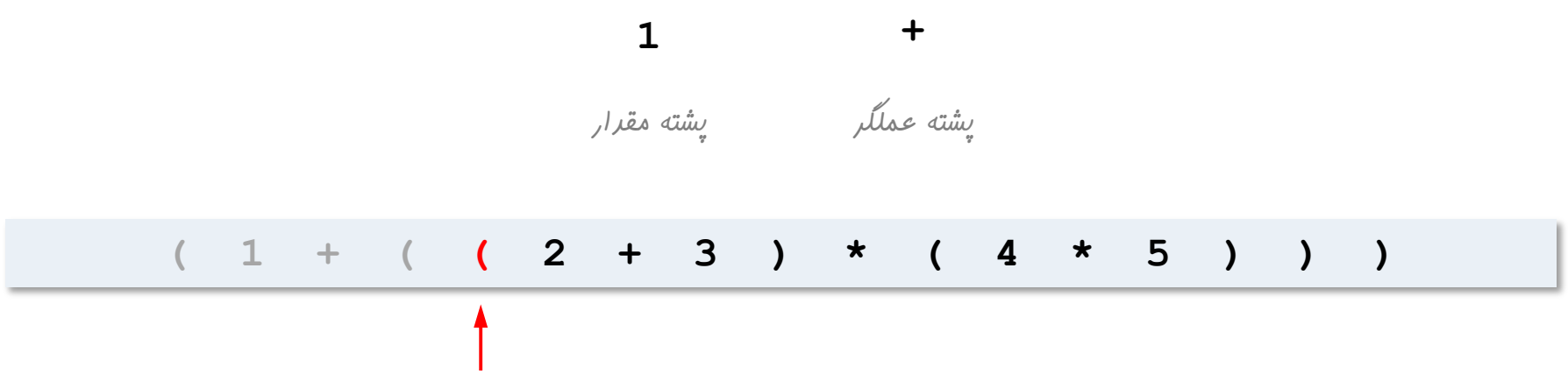


ارزیابی عبارت‌های میانوندی

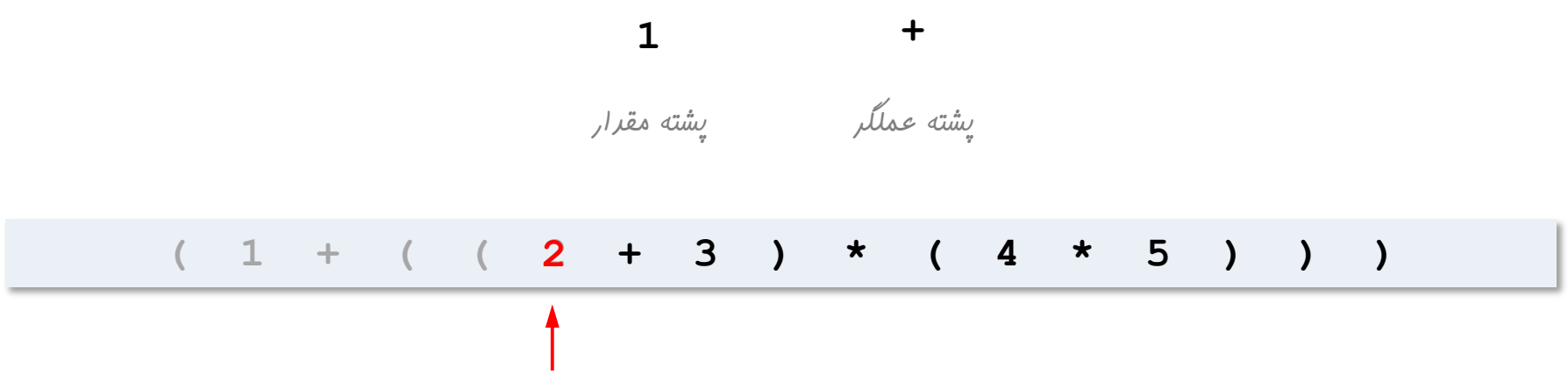
1	+
پشته مقدار	پشته عملگر



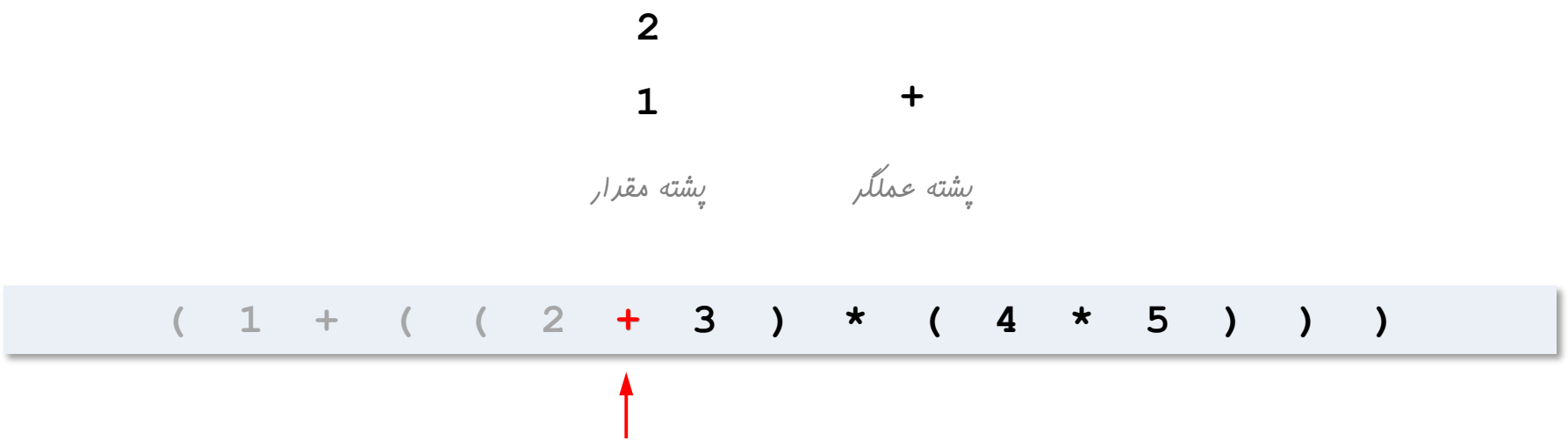
ارزیابی عبارت‌های میانوندی



ارزیابی عبارت‌های میانوندی



ارزیابی عبارت‌های میانوندی



ارزیابی عبارت‌های میانوندی

پشته عملگر	پشته مقدار
+	2
+	1

ارزیابی عبارت‌های میانوندی

پشته مقدار	پشته عملگر
3	
2	+
1	+

ارزیابی عبارت‌های میانوندی

پشته مقدار	پشته عملگر
3	
2	+
1	+

(1 + ((2 + 3) * (4 * 5)))

ارزیابی عبارت‌های میانوندی

2 + 3 = 5

+	1
پشته عملگر	پشته مقدار

(1 + ((2 + 3) * (4 * 5)))

ارزیابی عبارت‌های میانوندی

5

1

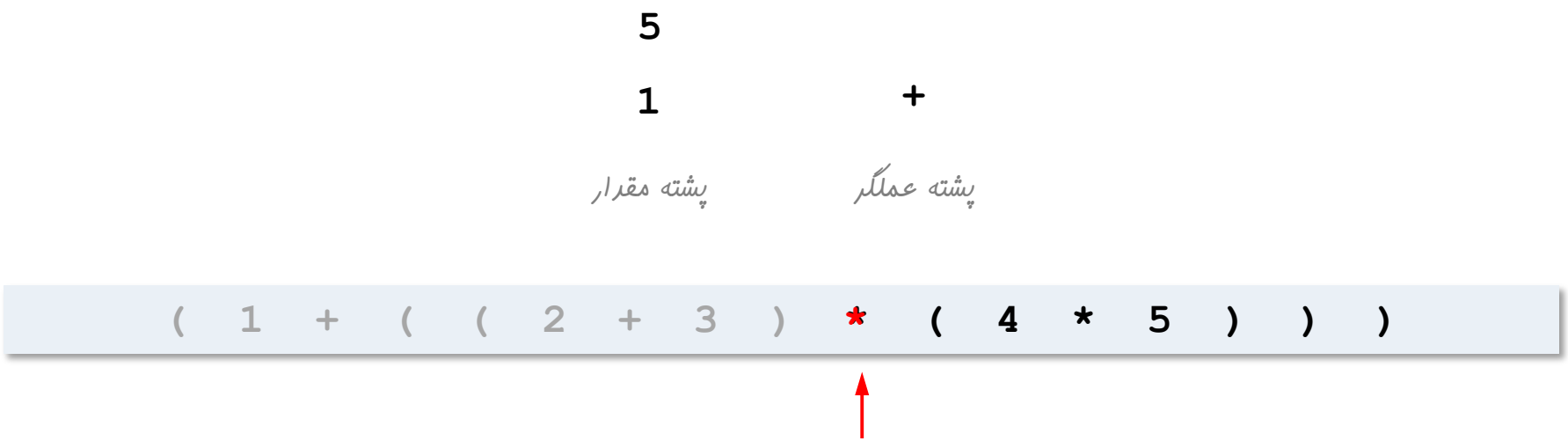
+

پشته مقدار

پشته عملگر

(1 + ((2 + 3) * (4 * 5)))

ارزیابی عبارت‌های میانوندی



ارزیابی عبارت‌های میانوندی

پشته عملگر	پشته مقدار
*	5
+	1

(1 + ((2 + 3) * (4 * 5)))

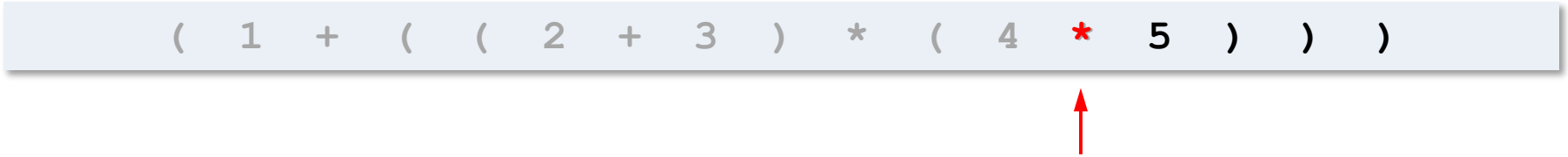
ارزیابی عبارت‌های میانوندی

پشته عملگر	پشته مقدار
*	5
+	1

(1 + ((2 + 3) * (4 * 5)))

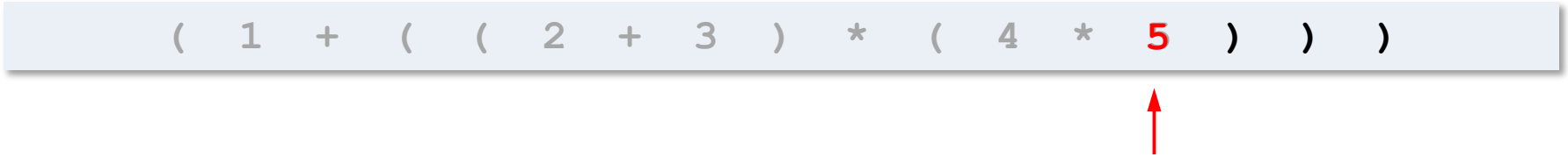
ارزیابی عبارت‌های میانوندی

پشته مقدار	پشته عملگر
4	
5	*
1	+



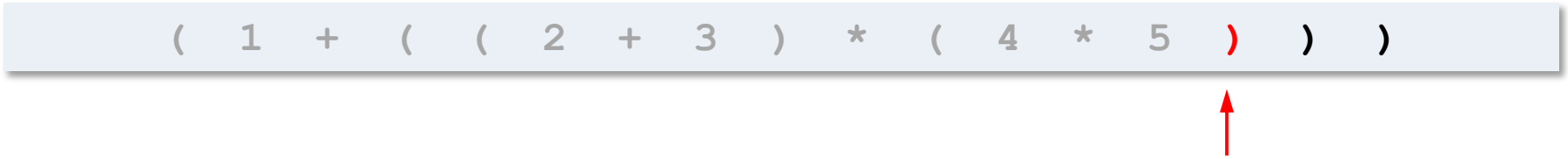
ارزیابی عبارت‌های میانوندی

پشته عملگر	پشته مقدار
*	4
*	5
+	1



ارزیابی عبارت‌های میانوندی

پشته مقدار	پشته عملگر
5	
4	*
5	*
1	+



ارزیابی عبارت‌های میانوندی

4 * 5 = 20

پشته عملگر	پشته مقدار
*	5
+	1

(1 + ((2 + 3) * (4 * 5)))

ارزیابی عبارت‌های میانوندی

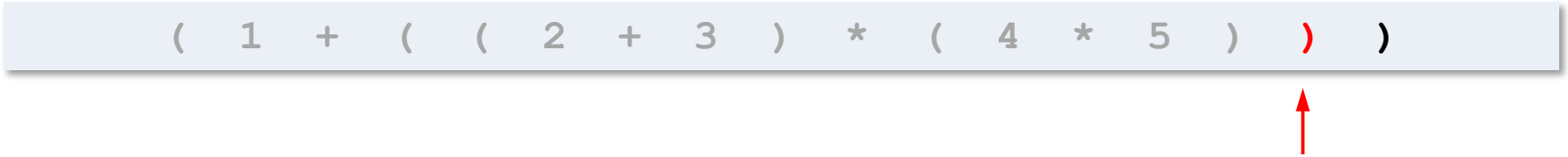
20

پشته عملگر	پشته مقدار
*	5
+	1

(1 + ((2 + 3) * (4 * 5)))

ارزیابی عبارت‌های میانوندی

پشته عملگر	پشته مقدار
+	1
*	5
	20



ارزیابی عبارت‌های میانوندی

5 * 20 = 100

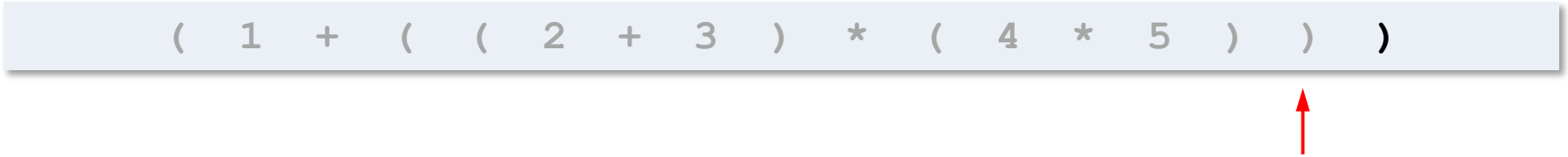
1	+
پشته مقدار	پشته عملگر

(1 + ((2 + 3) * (4 * 5)))

ارزیابی عبارت‌های میانوندی

100

1	+
پشته مقدار	پشته عملگر



ارزیابی عبارت‌های میانوندی

1 + 100 = 101

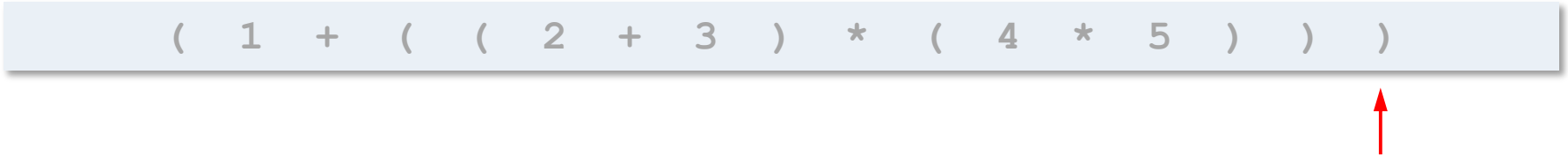
پشته عملگر	پشته مقدار
	100
+	1

(1 + ((2 + 3) * (4 * 5)))

ارزیابی عبارت‌های میانوندی


```
1  +  100  =  101
```

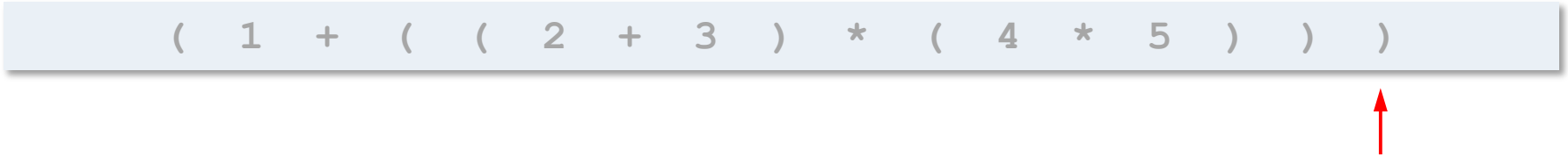
پشته عملگر پشته مقدار



ارزیابی عبارت‌های میانوندی

101

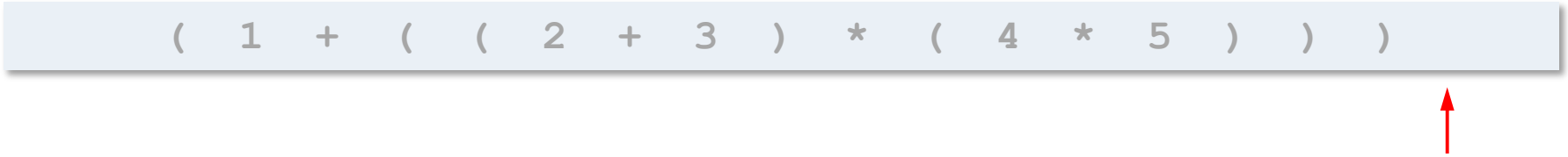
پشته عملگر پشته مقدار



ارزیابی عبارت‌های میانوندی

نتیجه **101**

پشته عملگر پشته مقدار



ارزیابی عبارت‌های میانوندی


```
public class Evaluate {
     public static void main(String[] args)  {
          Stack<String> ops  = new Stack<String>();
          Stack<Double> vals = new Stack<Double>();
          while (!StdIn.isEmpty()) {
               String s = StdIn.readString();
               if (s.equals("(")) ;
               else if (s.equals("+")) ops.push(s);
               else if (s.equals("*")) ops.push(s);
               else if (s.equals(")")) {
                    String op = ops.pop();
                    if      (op.equals("+")) vals.push(vals.pop() + vals.pop());
                    else if (op.equals("*")) vals.push(vals.pop() * vals.pop());
               }
               else vals.push(Double.parseDouble(s));
          }
          StdOut.println(vals.pop());
     }
}
```

ارزیابی عبارت‌های ریاضی

- مشاهده‌ی ۱. الگوریتم دو پشته‌ای برای عبارت‌های پسوندی نیز درست عمل می‌کند.

```
( 1 ( ( 2 3 + ) ( 4 5 * ) * ) + )
```

- مشاهده‌ی ۲. در شکل پسوندی (لهستانی معکوس) نیازی به وجود پرانتزها نیست.

```
1 2 3 + 4 5 * * +
```

کاربردهای صف

- کاربردهای آشنا.
  - لیست پخش در **iTunes**
  - بافرهای داده‌ای
  - انتقال داده‌ی غیرهمگام (ورودی خروجی فایل، سوکت‌ها)
  - پردازش درخواست‌ها در یک منبع اشتراکی (چاپگر، پردازنده)

- شبیه‌سازی دنیای واقعی.
  - تجزیه و تحلیل ترافیک
  - زمان انتظار مشتری‌ها در بانک
  - تعیین تعداد صندوق‌دارهای لازم در یک ابرفروشگاه