Pilas y Colas

Estructuras de Datos

1

Nuevas Estructuras

- Pila (Stack)
- Cola (Queue)
- Cola de Prioridad (Priority Queue)

- ¿Qué es lo "nuevo"?
  - Contraste con los arreglos
  - Uso
  - Acceso
  - Abstracción

2

Uso

- Los arreglos son propicios para las bases de datos
  - Los datos pueden ser accedidos y modificados
  - Operaciones fáciles para inserción, eliminación y búsqueda
    - Aunque algunos de estos consumen mucho tiempo
- Las pilas y las colas son buenas para desarrollar herramientas de programación
  - Los datos no serán tocados por el usuario
  - Se utilizan y luego se descartan

3

Acceso

- Las arreglos permiten un acceso inmediato a cualquier elemento
  - Toma tiempo constante
  - Muy rápido
- Las pilas y las colas sólo permiten el acceso a un elemento a la vez
  - Mucho más restringido

4

Abstracción

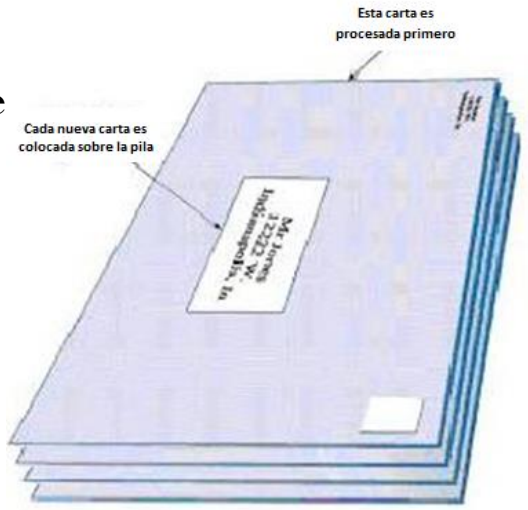
- Un poco mayor que los arreglos. ¿Por que?
  - Cuando un usuario indexa un arreglo, especifica una dirección de memoria
    - Indirectamente, porque hace:
    - Nombre Arreglo -> dirección del primer elemento
    - Índice -> desplazamiento de ese tamaño de la dirección * tamaño de la celda de un arreglo
  - Con pilas y colas, todo se hace a través de métodos
    - El usuario no tiene idea de lo que sucede detrás de la escena
    - Además, no se requiere de un tamaño inicial
  - Lo más importante
    - Pilas, colas y colas de prioridad pueden utilizar arreglos como su estructura subyacente
    - O vinculado con listas enlazadas...
    - Desde la perspectiva del usuario, son una y la misma

5

Pila

- **Una pila sólo permite el acceso al último ítem insertado**
- **Para acceder al siguiente, se deben remover los anteriores**
- **Analogía: Correos de Chile**

Si recibimos un atado de cartas, típicamente abrimos la primera que esta a la vista, pagamos por su recepción al cartero, nos deshacemos de ella y abrimos la segunda.



6

Implicación de Rendimiento

- ¡Note lo que ya podemos inferir sobre del rendimiento de pila!
- Es muy importante que seamos capaces de procesar el correo de manera eficiente
- De lo contrario, ¿que pasaría con las cartas que están al fondo?

- En otras palabras, ¿qué sucede con las cuentas por pagar si nunca llegamos a ellas? ☺

- Una pila es lo que se conoce como una estructura LIFO (last-in first-out) " la última en entrar, es la primera en salir"
  - Sólo podemos insertar en la parte superior (push)
  - Sólo podemos acceder al elemento en la parte superior (echar un vistazo/peek)
  - Sólo podemos eliminar de la parte superior (pop)

7

Aplicaciones

- **Compiladores**
  - **Balanceo de paréntesis redondos, cuadrados**
  - **Tablas de Símbolos**
- **Análisis (parsing) de expresiones aritméticas**
- **Navegación de los nodos de árboles y grafos**
- **Invocación métodos**
- **calculadoras de bolsillo**

8

La Operación 'push'

- **Push (Empujar) implica la colocación de un elemento en la parte superior de la pila**
- **Analogía: Un día de trabajo**
  - **Te asignan un proyecto a largo plazo (push)**
  - **Usted es destinado a cooperar temporalmente con el proyecto B (push)**
  - **Desde contabilidad le solicitan asistir a una reunión sobre el proyecto de C (push)**
  - **Recibe una llamada de emergencia para obtener ayuda sobre proyecto D (push)**
- **En cualquier momento, usted está trabajando en el proyecto más recientemente asignado ("empujado")**

9

La operación 'pop'

- **Pop consiste en extraer el elemento superior de la pila**
- **Analogía: Día de trabajo**
  - **Termina la llamada de emergencia con el proyecto D (pop)**
  - **Termina la reunión del proyecto de C (pop)**
  - **Termine la ayuda sobre el proyecto B (pop)**
  - **Completar el proyecto de largo plazo A (pop)**
- **Cuando todo es extraído de la pila, se considera que esta es una *pila vacía***
- **Las pilas siempre están inicialmente vacías**

10

La operación 'peek'

- **Peek le permite ver el elemento en la parte superior de la pila sin eliminarlo.**

- **Nota al margen: Los tamaños de las pilas a menudo no son demasiado grandes**
- **Esto es porque en la mayoría de las aplicaciones en las que se utilizan pilas, basta con descartar los datos después de usarlos**

11

Clase Stack

- **Implementación en Java**

- **Revisémosla**
- **Tenga en cuenta, que tenemos que elegir nuestra estructura de datos interna**
  - **Por ahora nos quedaremos con lo que sabemos: Un arreglo**
- **Y analizaremos el main ()**

12

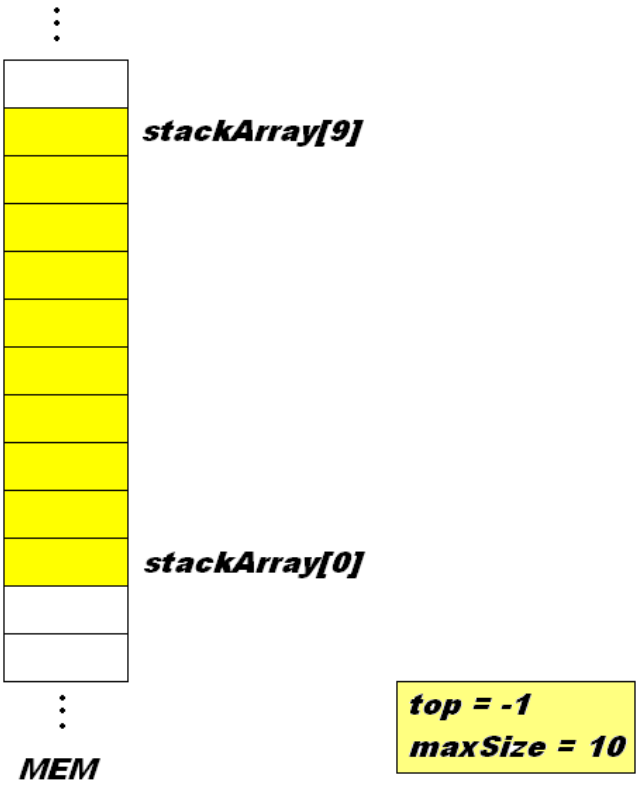
Métodos de la clase Stack

- **Constructor:**
  - **Acepta un tamaño, crea una nueva pila**
  - **Internamente asigna un arreglo con esa cantidad de slots**
- **push()**
  - **Incrementa top y almacena un ítem de datos allí**
- **pop()**
  - **Retorna el valor en el tope y decrece top**
  - **Note que el valor permanece en el arreglo! Solo se hace que sea inaccesible (¿por qué?)**
- **peek()**
  - **Retorna el valor en el tope de la pila sin modificarla**
- **isFull(), isEmpty()**
  - **Retorna verdadero (true) o falso (false)**

13

Gráficamente, veamos la ejecución de main()

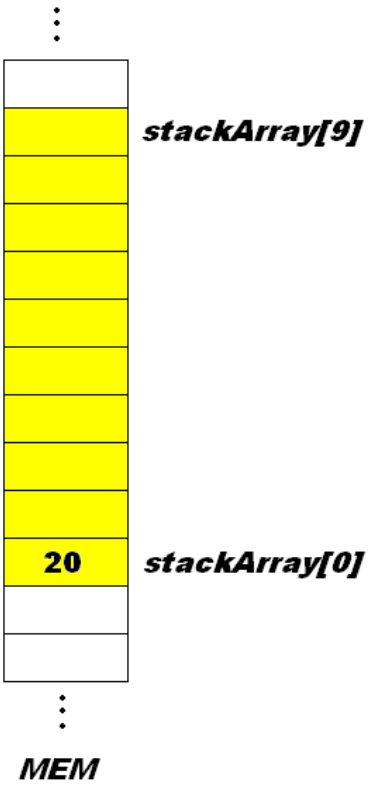
- **StackX theStack =**
  new StackX(10);



14

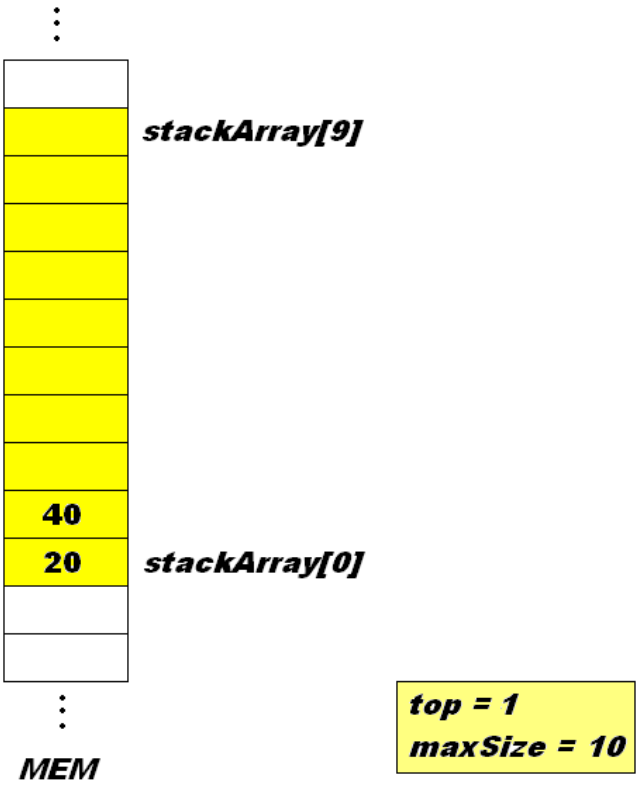
Push

- theStack.push(20);
- top es incrementado
- 20 es almacenado en el slot con índice top



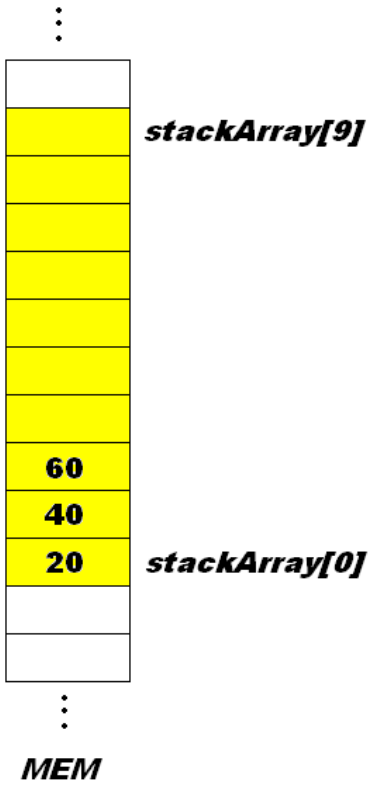
15

- theStack.push(40);
- top es incrementado
- 40 es almacenado en el slot con índice top



16

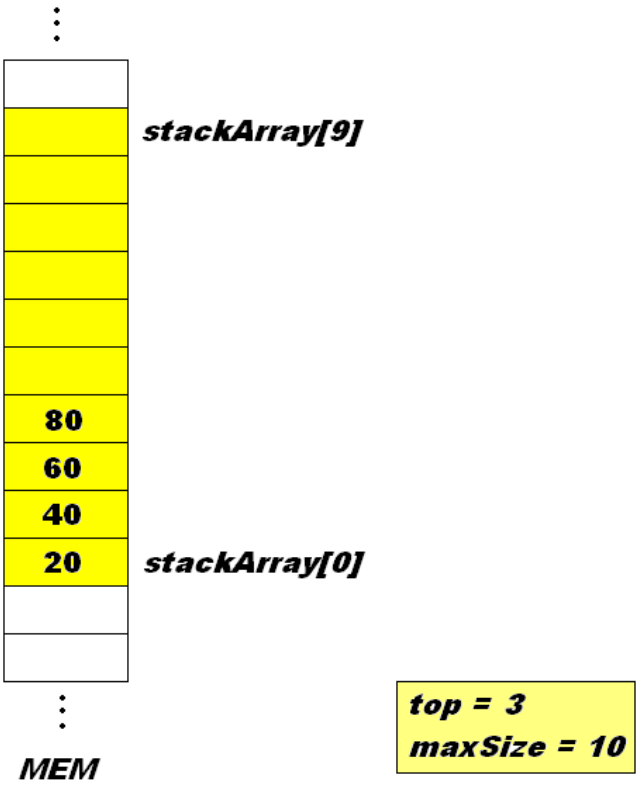
- theStack.push(60);
- top es incrementado
- 60 es almacenado en el slot con índice top



top = 2
maxSize = 10

17

- theStack.push(80);
- top es incrementado
- 80 es almacenado en el slot con índice top

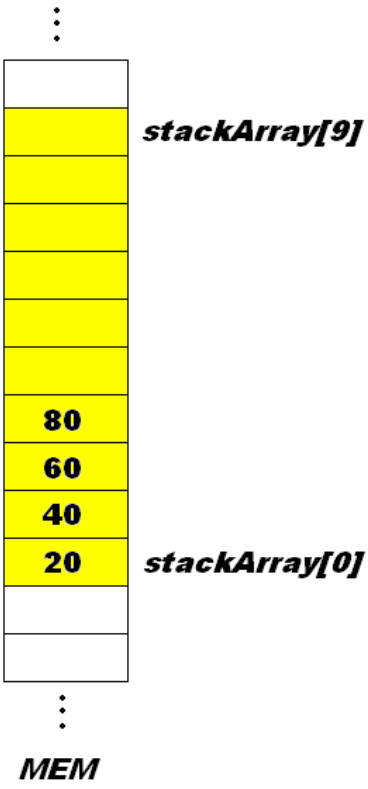


18

Pop

- while (!theStack.isEmpty())
  {
  long value = theStack.pop()
  …

- El elemento indexado por **top** es asignado a **value**
- **top** es reducido en 1



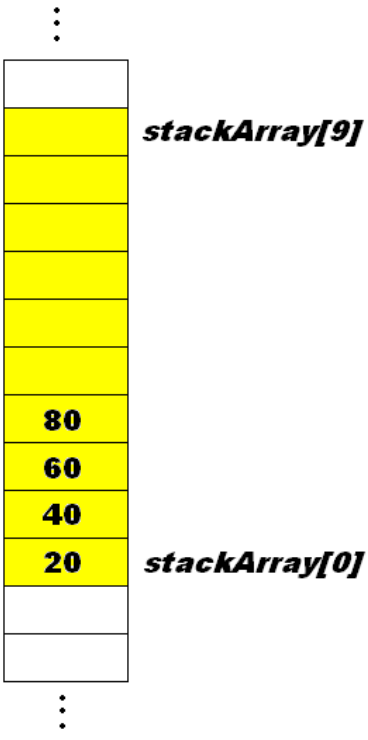
value = 80
top = 2
maxSize = 10

19

Print

- while (!theStack.isEmpty())
  {
  …
  System.out.print(value)
  System.out.print(“”)

PANTALLA



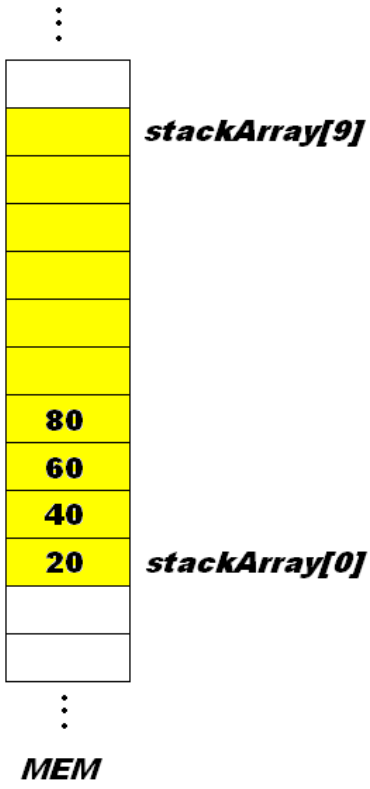
MEM

value = 80
top = 2
maxSize = 10

20

Pop

- while (!theStack.isEmpty())
  {
  long value = theStack.pop()
  …
- El elemento indexado por **top** es asignado a **value**
- **top** es reducido en 1



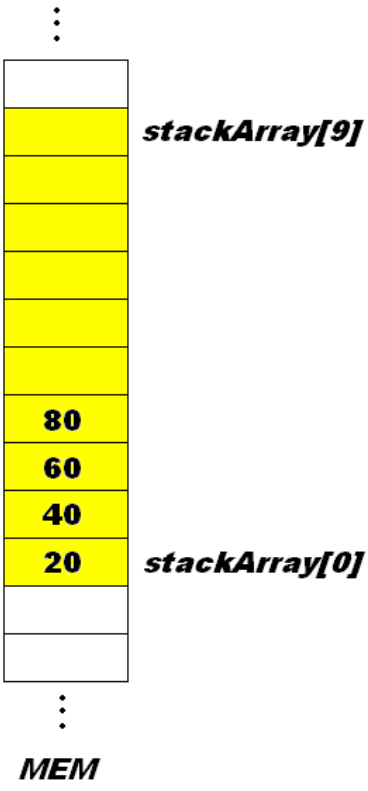
value = 60
top = 1
maxSize = 10

21

Print

- while (!theStack.isEmpty())
  {
  …
  System.out.print(value)
  System.out.print(“”)

PANTALLA



MEM

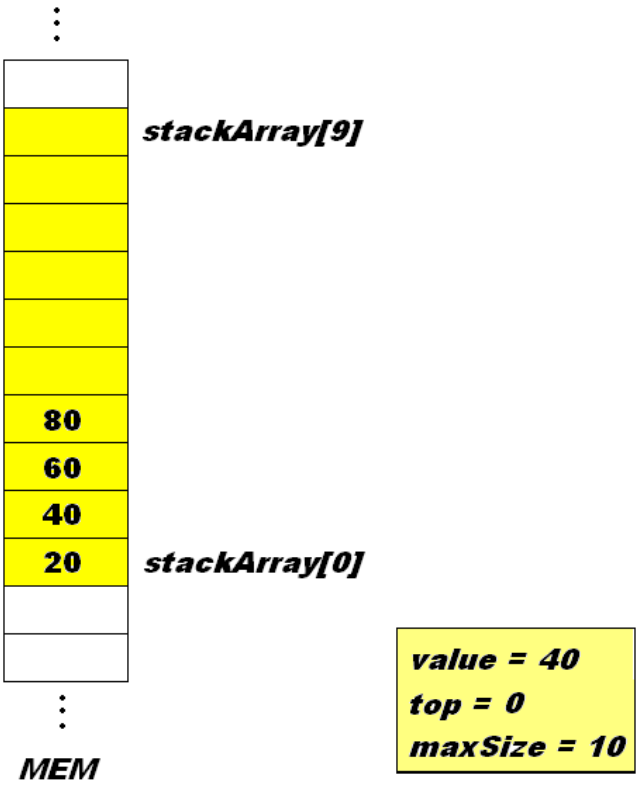
value = 60
top = 1
maxSize = 10

22

Pop

- while (!theStack.isEmpty())
  {
    long value = theStack.pop()
    …

- El elemento indexado por **top** es asignado a **value**
- **top** es reducido en 1

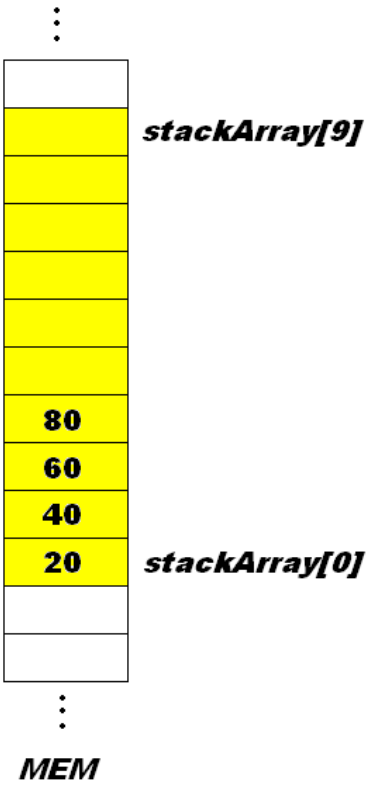


23

Print

- while (!theStack.isEmpty())
  {
  …
  System.out.print(value)
  System.out.print(“”)

PANTALLA

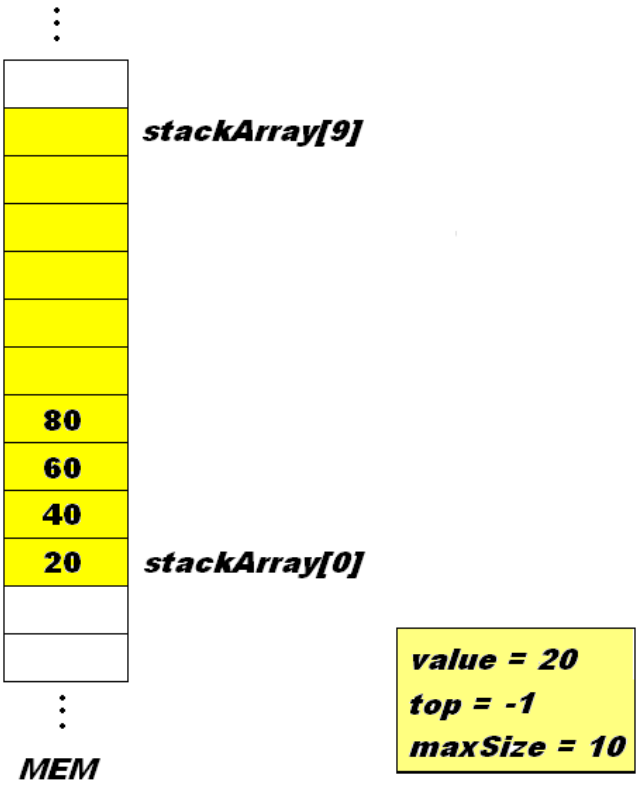


value = 40
top = 0
maxSize = 10

24

Pop

- while (!theStack.isEmpty())
  {
  long value = theStack.pop()
  …
- El elemento indexado por **top** es asignado a **value**
- **top** es reducido en 1

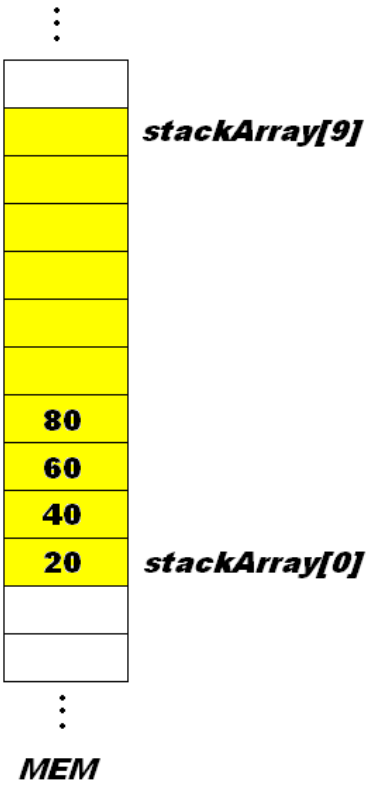


25

Print

- while (!theStack.isEmpty())
  {
  …
  System.out.print(value)
  System.out.print(“”)

PANTALLA



value = 20
top = -1
maxSize = 10

26

Manejo de Errores

- ¿Cuándo deberíamos realizar el control de errores en el caso de la pila?
- ¿Qué función debería realizarlo?
- ¿Y cómo lo haríamos?

27

Ejemplo de Aplicación: Invertir una Palabra

- Vamos a utilizar una pila para tomar un string e invertir sus caracteres
  - ¿Cómo podría funcionar esto? Veámoslo.
- Recuerda algunas operaciones disponibles para manejar Strings:
- Si tenemos un string s
  - s.charAt(j) <- Retorna el caracter con índice j
  - s + "…" <- Agrega un string (o caracter a s)
  - ¿Qué necesitamos cambiar de nuestra clases Stack?
- Implementación en Java clase Reverser

28

Ejemplo de aplicación: Chequeo de Paréntesis

- ¡Esto lo hacen los compiladores!
- Analizan (Parse) líneas de código (strings) de un lenguaje de programación
- Muestra de paréntesis en Java:
  - {, }
  - [,]
  - (,)

- Todos los paréntesis de apertura deben ser igualados por los de cierre
- Además, los paréntesis de apertura posteriores deben estar más cerca que los anteriores
  - Veamos cómo una la pila puede ayudarnos aquí?

29

Ejemplo de Strings

- c[d]
- a{b[c]d}e
- a{b(c]d}e
- a[b{c}d]e}
- a{b(c)

- ¿Cuáles de estos son correctos?
- ¿Cuáles son incorrectos?

30

Algoritmo

- Lea cada un carácter a la vez
- Si es un paréntesis de apertura, colóquelo en la pila
- Si es un paréntesis de cierre, ejecute un pop en la pila
  - Si la pila está vacía, imprima error
  - De lo contrario, si es un paréntesis de apertura y calza, continúe
  - De lo contrario, imprima error
- Si la pila no está vacía al final, imprima error

31

Ejemplo

- **Veamos la pila para a{b(c[d]e)f}**

Caracter	Pila	Acción
a		x
{	{	push '{'
b	{	x
(	{(	push '('
c	{(	x
[	{([	push '['
d	{([	x
]	{(	pop '[', calce
e	{(	x
)	{	pop '(', calce
f	{	x
}		pop '{', calce

Ejemplo

- Verifique uno con errores: a[b{c}d]e}
- Realice el ejercicio en su cuaderno

33

Implementación en Java: Tarea

- **Implemente el analizador**
- **Escriba una función que acepte un string como entrada**
- **Y retorne true o false dependiendo si el string tiene delimitadores que calcen**

- **Podemos usar la clase Stack en cuyo arreglo interno se almacenaran los caracteres**

34

Pilas: Evaluación

- **Para las herramientas que vimos: invertir una palabra y delimitadores de calce, ¿considera que las pilas nos ayudan a hacer las tarea con mayor facilidad?**
  - **es decir, ¿Hubiese ido más difícil con arreglos?**
  - **¿Por qué utilizar una pila en su programa hace que sea más fácil de entenderlo?**
- **Eficiencia**
  - **Push -> O(1) (Inserción es rápida, pero sólo en el tope)**
  - **Pop -> O(1) (Eliminación es rápida, pero sólo en el tope)**
  - **Peek -> O(1) (Acceso es rápido, pero sólo en el tope)**

35

Colas

- **Sinónimos: Línea, fila**
- **Se parece a una pila**
  - **Excepto, el primero que entra es el primero que sale**
  - **Por lo tanto es una estructura FIFO (First-in First-Out).**

36

Analogía:

- **Fila en la entrada de un cine**
- **La ultima persona en la línea es la última en comprar su ticket**

37

Aplicaciones

- Búsqueda en grafos
- Simulación de situaciones del mundo real
  - Gente que espera en colas bancarias
  - Aviones a la espera para despegar
  - Paquetes en espera para ser transmitidos a través de la Internet
- Hardware
  - Cola de una impresora
  - Golpes de un teclado
    - Garantiza el orden de procesamiento correcto

38

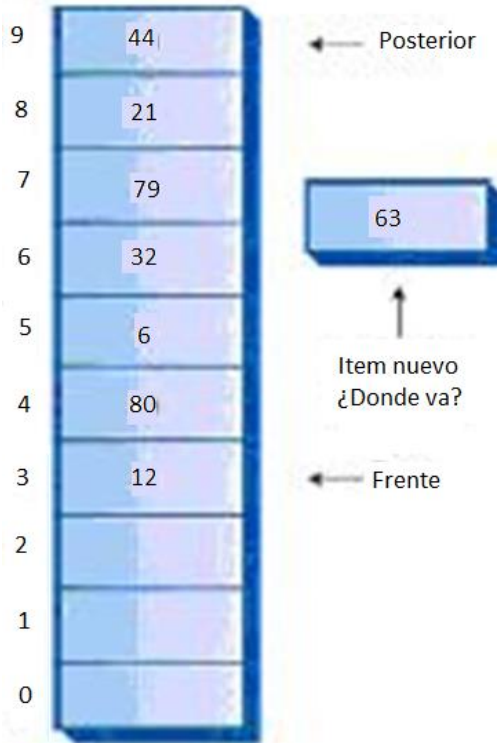
Operaciones de Colas

- **insert() (insertar)**
  - **También conocidos como put (), add (), o enqueue ()**
  - **Inserta un elemento en la parte posterior de la cola**
- **remove() (eliminar)**
  - **También conocido como get(), delete(), or dequeue()**
  - **Remueve un elemento de la parte frontal de la cola**
- **peekRear() (mirar final)**
  - **Elemento al final de la cola**
- **peekFront() (mirar el frente)**
  - **Elemento al frente de la cola**

39

¡¡Insertar y eliminar: ocurren en los extremos!!!

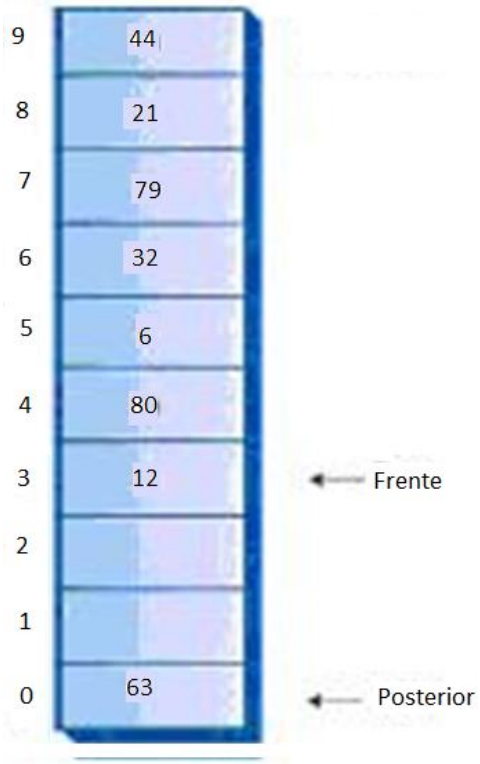
- **En una pila, estas operaciones se producen en el mismo extremo**
  - **Eso significa que si eliminamos un elemento podemos reutilizar su ranura**
- **En una cola, no podemos hacer esto**
- **A menos que ....**



40

Cola Circular

- **Índices ‘ se cruzan’**



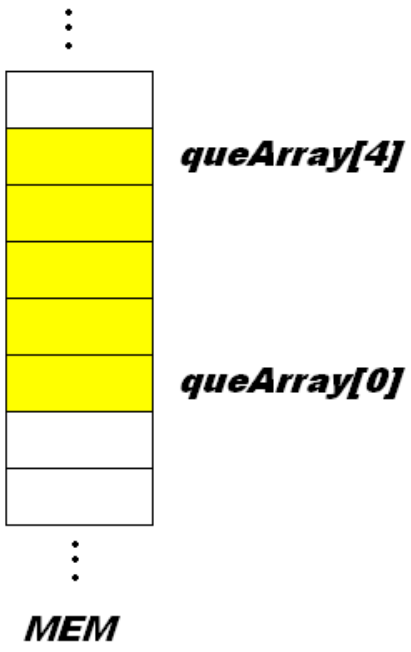
41

Implementación Java: Tarea

- **Implementación propuesta usa nuevamente un arreglo interno**
- **Construiremos esta clase**
- **Analizaremos la función main gráficamente**

42

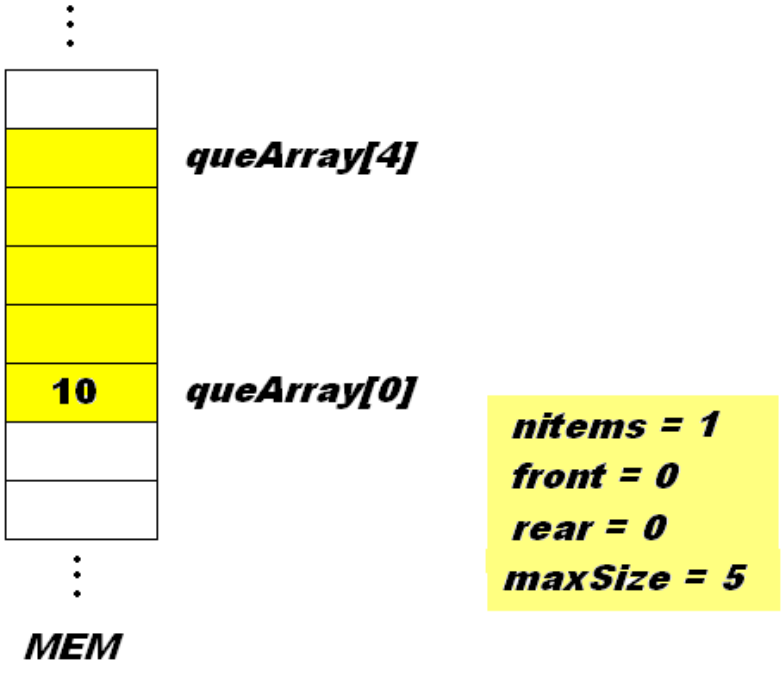
Queue theQueue = new Queue(5);



nitems = 0
front = 0
rear = -1
maxSize = 5

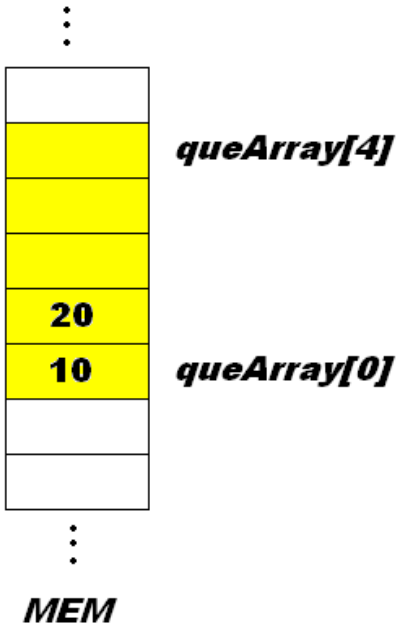
43

theQueue.insert(10);



44

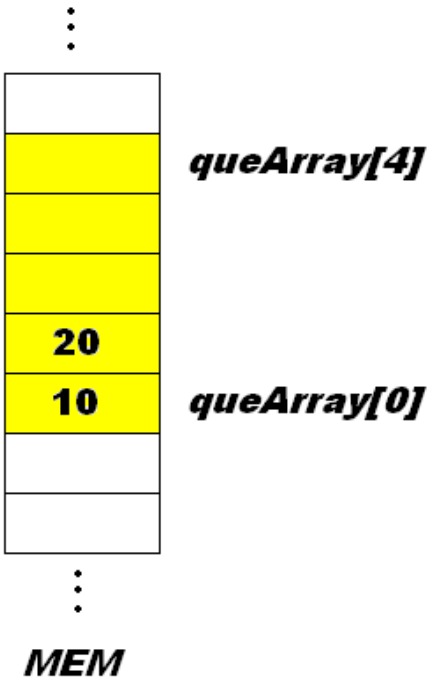
theQueue.insert(20);



nitems = 2
front = 0
rear = 1
maxSize = 5

45

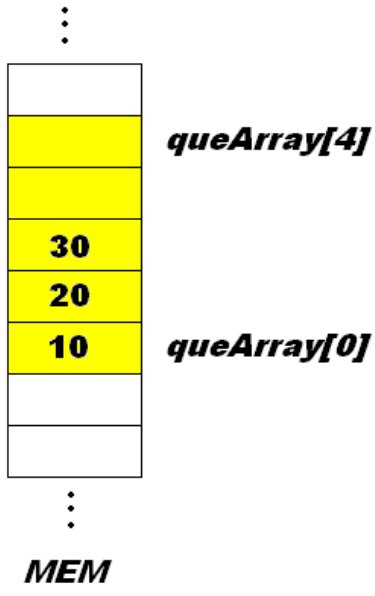
theQueue.insert(30);



nitems = 2
front = 0
rear = 1
maxSize = 5

46

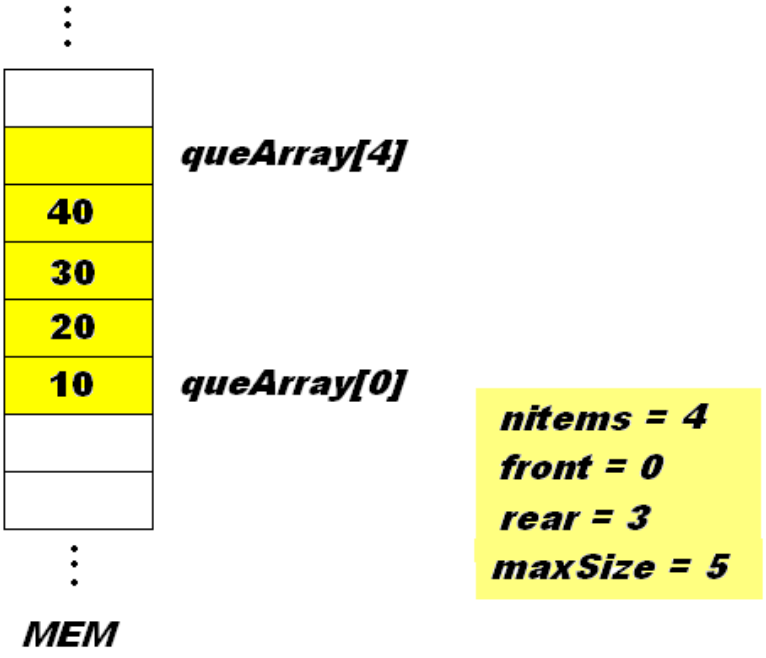
theQueue.insert(30);



nitems = 3
front = 0
rear = 2
maxSize = 5

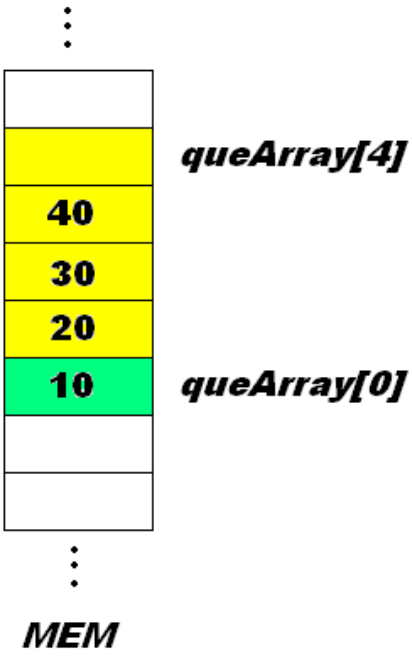
47

theQueue.insert(40);



48

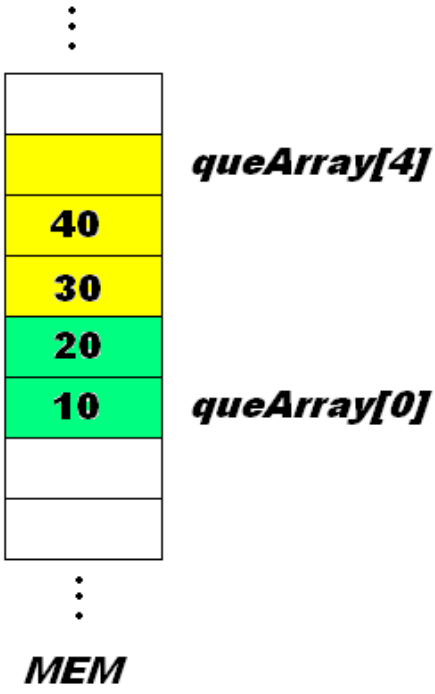
theQueue.remove();



nitems = 3
front = 1
rear = 3
maxSize = 5

49

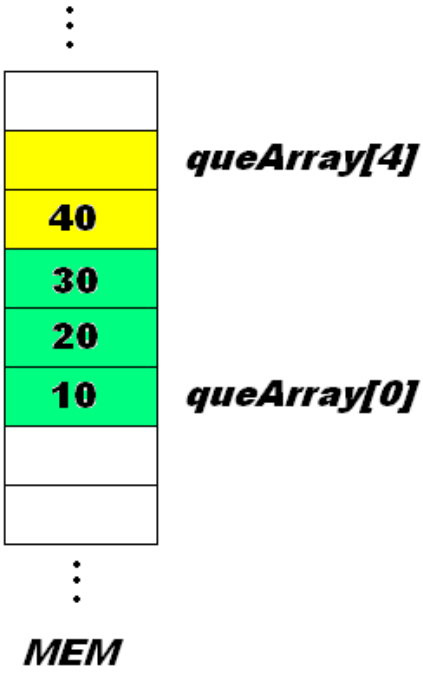
theQueue.remove();



nitems = 2
front = 2
rear = 3
maxSize = 5

50

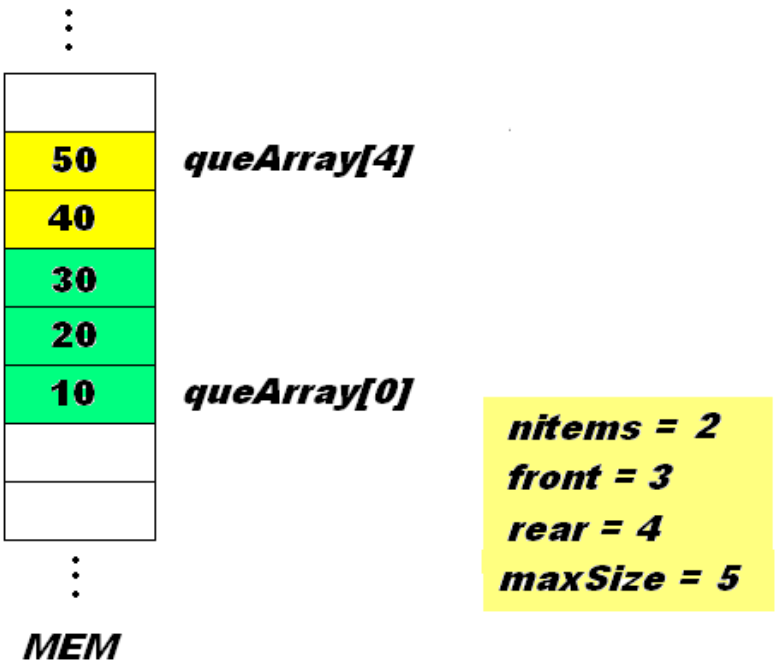
theQueue.remove();



nitems = 1
front = 3
rear = 3
maxSize = 5

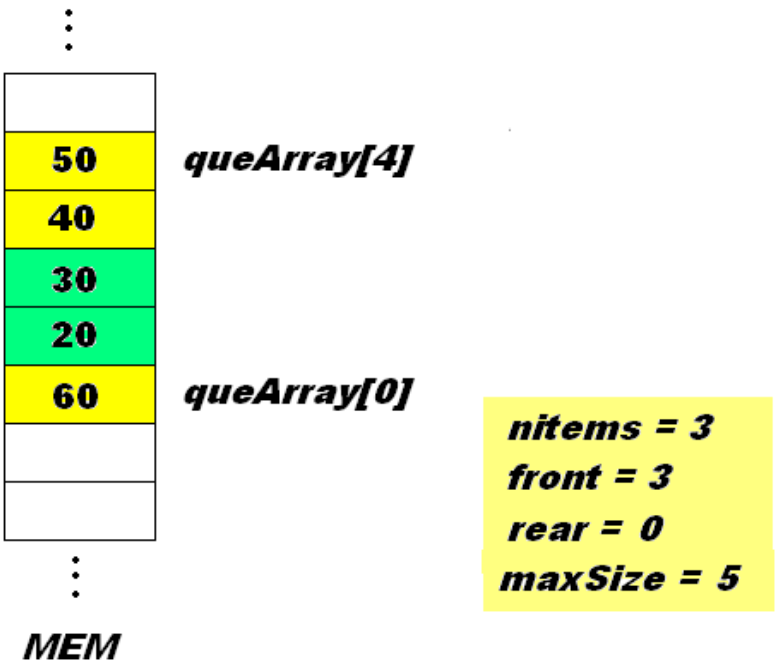
51

theQueue.insert(50);



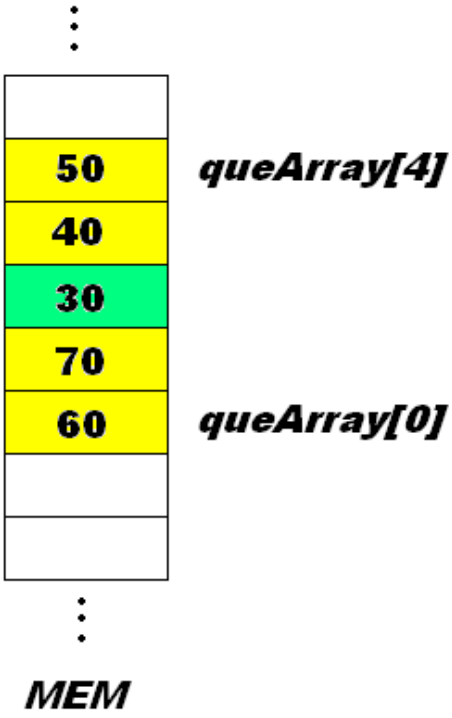
52

theQueue.insert(60);



53

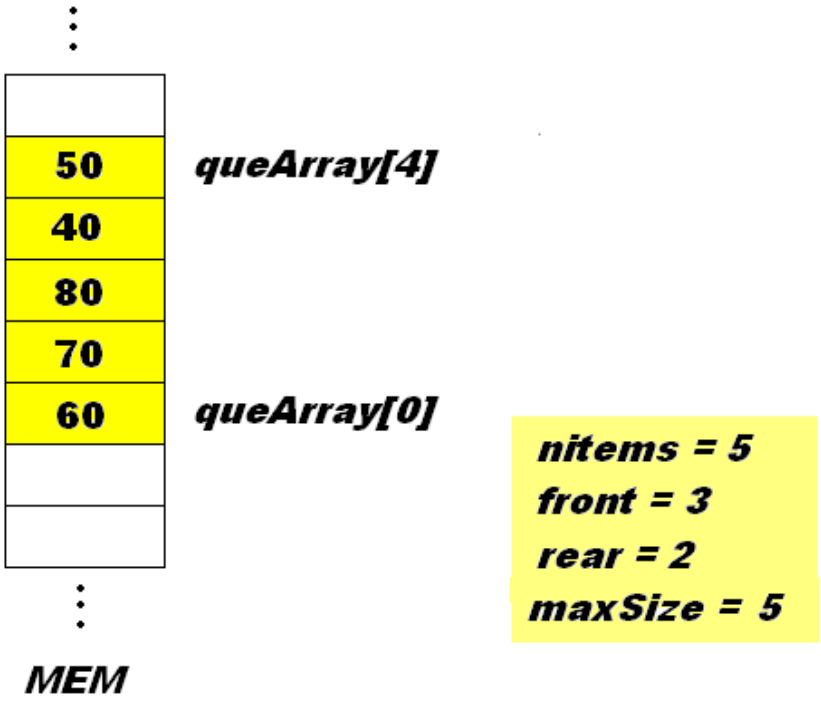
theQueue.insert(70);



nitems = 4
front = 3
rear = 1
maxSize = 5

54

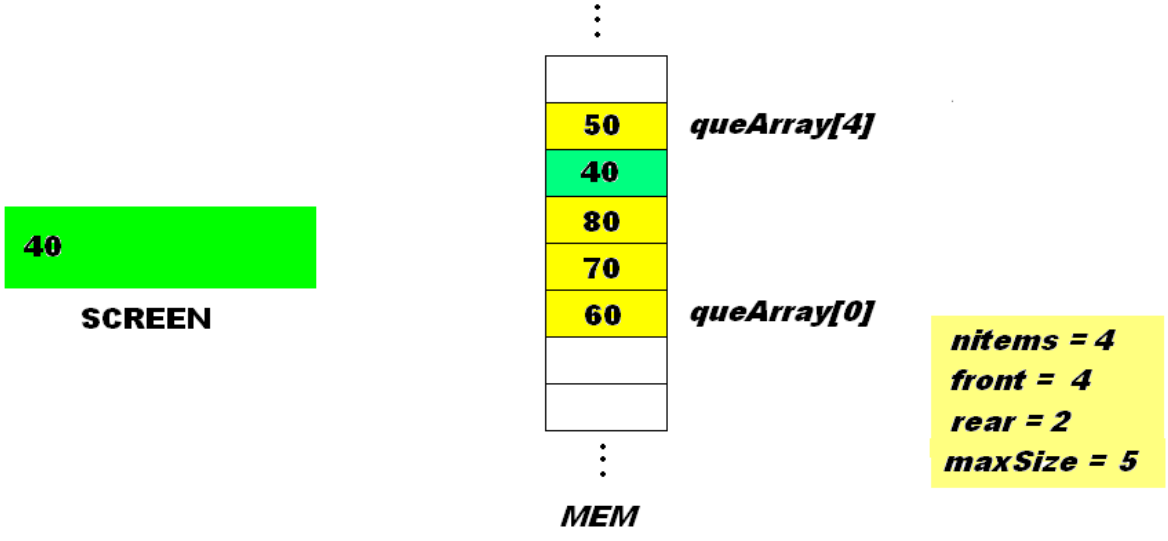
theQueue.insert(80);



55

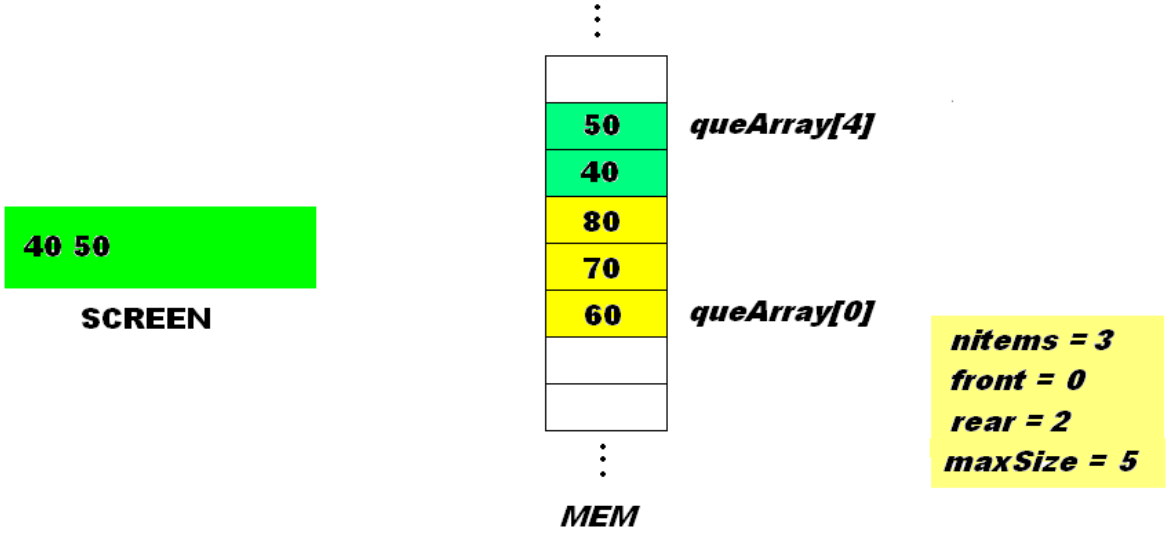
Remover e imprimir...

- while (!theQueue.isEmpty())
- long n = theQueue.remove();
- System.out.print(n);



Remover e imprimir...

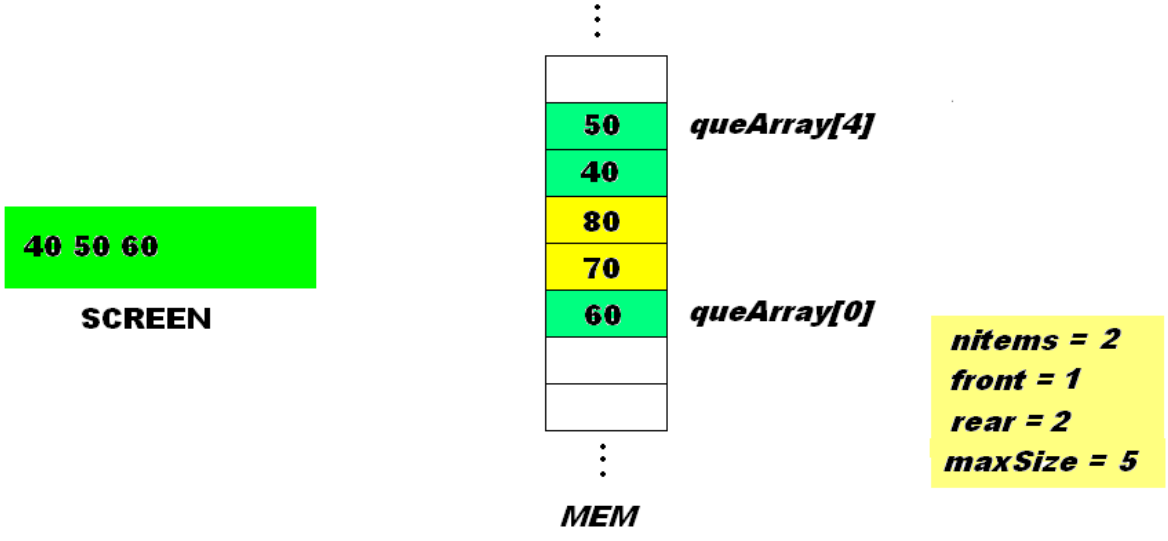
- while (!theQueue.isEmpty())
- long n = theQueue.remove();
- System.out.print(n);



57

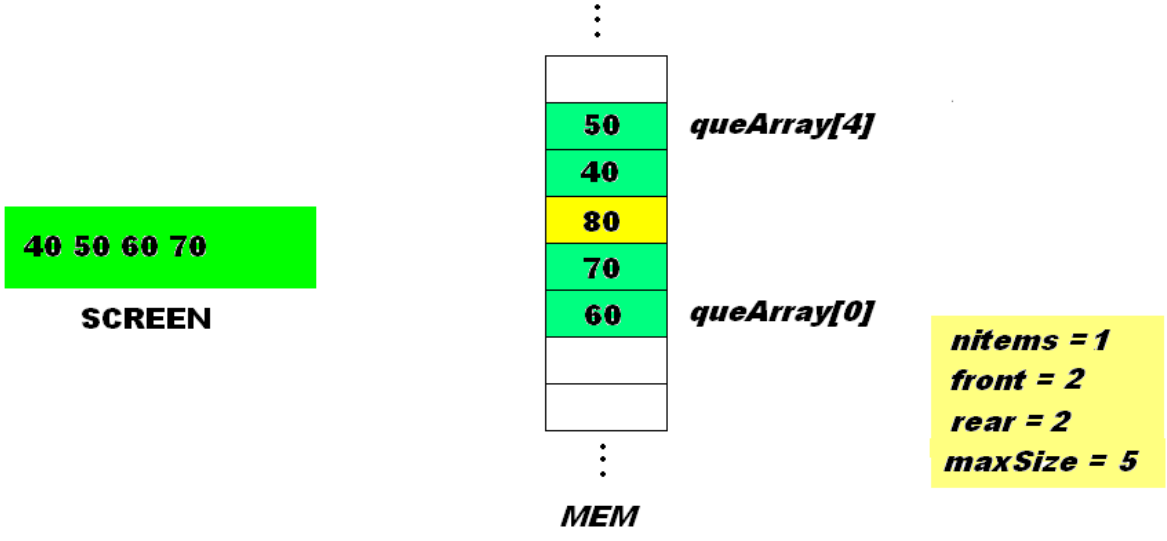
Remover e imprimir...

- while (!theQueue.isEmpty())
- long n = theQueue.remove();
- System.out.print(n);



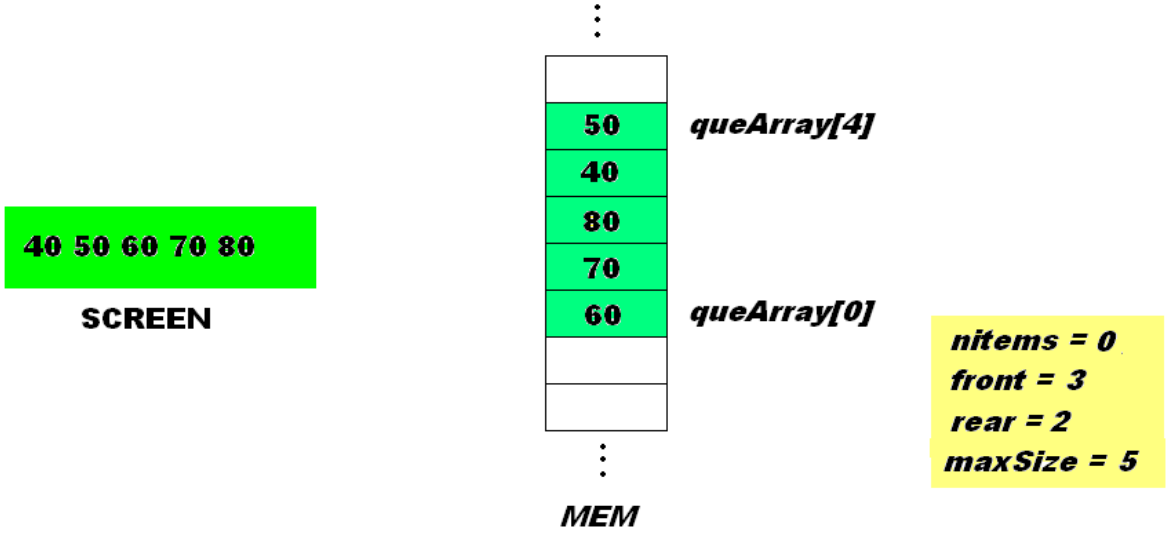
Remover e imprimir...

- while (!theQueue.isEmpty())
- long n = theQueue.remove();
- System.out.print(n);



Remover e imprimir...

- while (!theQueue.isEmpty())
- long n = theQueue.remove();
- System.out.print(n);



Colas: Evaluación

- Algunas implementaciones remueven nItems
  - Estas usan los índices posterior y frontal para determinar si la cola esta llena , vacía o su tamaño
- Eficiencia
  - Igual que una pila:
    - Push: O(1) sólo en la parte posterior
    - Pop: O(1) sólo al frente
    - Acceso: O(1) sólo al frente

61

Colas de Prioridad

- Al igual que una Cola
  - Tiene un frente y una parte trasera
  - Los ítems se retiran desde la parte delantera
- Diferencia
  - Ya no es FIFO
  - Los ítems son ordenados
- Ya vimos arreglos ordenados. Una cola de prioridad es esencialmente una 'cola ordenada'
- Analogía con el Correo: deseas responder primero las cartas más importantes

62

Implementación de una Cola de Prioridad

- **Casi NUNCA usan arreglos. ¿Por qué?**
- **Usualmente emplean un heap (lo estudiaremos después)**
- **Aplicaciones en computación**
  - **Programas con mayor prioridad se ejecutan primero**
  - **Impresión de jobs se ordena por prioridad**
- **Buena característica: El mínimo o máximo ítem se puede encontrar en un tiempo O (1)**

63

Implementación en Java: Tarea

- **Realizar en Ayudantía**
- **La gran diferencia es en la función insert()**

- **Análisis**
  - **delete() - O(1)**
  - **insert() - O(n) (nuevamente, si usamos arreglos)**
  - **findMin() - O(1) si se ordena de forma ascendente**
  - **findMax() – O(1) si se ordena de forma descendente**

64