Programación de Computadores: Java - Memoria, Objetos y Clases Empaquetadoras (Wrappers)

Juan F. Pérez

Departamento MACC
Matemáticas Aplicadas y Ciencias de la Computación
Universidad del Rosario

juanferna.perez@urosario.edu.co

Segundo Semestre de 2017

Contenidos

Un Ejemplo

Un Ejemplo

Empecemos revisando el proyecto `proyectoDocumentos`

- Clases
- Atributos
- Constructores
- Otros Métodos

Memoria

Almacenando información

Unidades de almacenamiento:

Almacenando información

Unidades de almacenamiento:

- bit: 0 o 1

Almacenando información

Unidades de almacenamiento:

- bit: 0 o 1
- byte: 8 bits

Almacenando información

Unidades de almacenamiento:

- bit: 0 o 1
- byte: 8 bits
- palabra (word): 4 bytes

Almacenando información

Unidades de almacenamiento:

- bit: 0 o 1
- byte: 8 bits
- palabra (word): 4 bytes
- `int` y `float`: 32 bits

Almacenando información

Unidades de almacenamiento:

- bit: 0 o 1
- byte: 8 bits
- palabra (word): 4 bytes
- `int` y `float`: 32 bits
- `long` y `double`: 64 bits

Prefijos

Prefijos reflejan potencias de 2 más cercanas a la potencia de 10:

Prefijos

Prefijos reflejan potencias de 2 más cercanas a la potencia de 10:

kilo (K): $2^{10} = 1{,}024$

Prefijos

Prefijos reflejan potencias de 2 más cercanas a la potencia de 10:

kilo (K): $2^{10} = 1{,}024$

mega (M): $2^{20} = 1{,}048{,}576$

Prefijos

Prefijos reflejan potencias de 2 más cercanas a la potencia de 10:

- kilo (K): $2^{10} = 1{,}024$
- mega (M): $2^{20} = 1{,}048{,}576$
- giga (G): $2^{30} = 1{,}073{,}741{,}824$

Prefijos

Prefijos reflejan potencias de 2 más cercanas a la potencia de 10:

- kilo (K): $2^{10} = 1{,}024$
- mega (M): $2^{20} = 1{,}048{,}576$
- giga (G): $2^{30} = 1{,}073{,}741{,}824$
- tera (T): $2^{40} = 1{,}099{,}511{,}627{,}776$

Prefijos

Prefijos reflejan potencias de 2 más cercanas a la potencia de 10:

- kilo (K): $2^{10} = 1{,}024$
- mega (M): $2^{20} = 1{,}048{,}576$
- giga (G): $2^{30} = 1{,}073{,}741{,}824$
- tera (T): $2^{40} = 1{,}099{,}511{,}627{,}776$

$128 \text{ GB} = 128 * 1{,}073{,}741{,}824 = 137{,}438{,}953{,}472 \text{ bits}$

Números Binarios

Representación exacta de enteros agrupando bits:

1 bit: 0 o 1

Números Binarios

Representación exacta de enteros agrupando bits:

1 bit: 0 o 1

2 bits

$$00 = 0 * 2^1 + 0 * 2^0 = 0$$
$$01 = 0 * 2^1 + 1 * 2^0 = 1$$
$$10 = 1 * 2^1 + 0 * 2^0 = 2$$
$$11 = 1 * 2^1 + 1 * 2^0 = 3$$

Números Binarios

Representación exacta de enteros agrupando bits:

1 bit: 0 o 1

2 bits

$$00 = 0 * 2^1 + 0 * 2^0 = 0$$
$$01 = 0 * 2^1 + 1 * 2^0 = 1$$
$$10 = 1 * 2^1 + 0 * 2^0 = 2$$
$$11 = 1 * 2^1 + 1 * 2^0 = 3$$

4 bits

$$1111 = 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 15$$

Números Binarios

Representación exacta de enteros agrupando bits:

1 bit: 0 o 1

2 bits

$$00 = 0 * 2^1 + 0 * 2^0 = 0$$
$$01 = 0 * 2^1 + 1 * 2^0 = 1$$
$$10 = 1 * 2^1 + 0 * 2^0 = 2$$
$$11 = 1 * 2^1 + 1 * 2^0 = 3$$

4 bits

$$1111 = 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 15$$

8 bits (1 byte)

$$11111111 =$$
$$1*2^7+1*2^6+1*2^5+1*2^4+1*2^3+1*2^2+1*2^1+1*2^0 = 255$$

Números Hexadecimales

Base $16 = 2^4$ (4 bits)

Representación más compacta de número binarios

Números Hexadecimales

Base 16 $= 2^4$ (4 bits)

Representación más compacta de número binarios

Decimal	Hexadecimal	Decimal	Hexadecimal
0	0	8	8
1	1	9	9
2	2	10	A
3	3	11	B
4	4	12	C
5	5	13	D
6	6	14	E
7	7	15	F

Números Hexadecimales

Ejemplos

8 bits (1 byte):

$$1111\ 1111 = FF = 255$$
$$0011\ 1011 = 3B = 59$$
$$1101\ 1001 = D9 = 217$$

Direcciones en Memoria

Asigna números a cada *byte*

Direcciones en Memoria

Asigna números a cada *byte*

Ejemplo: memoria de 64 KB (65.535 bytes)

Del 0000

Al $FFFF = 2^16 - 1 = 65{,}535$

Direcciones en Memoria

Asigna números a cada *byte*

Ejemplo: memoria de 64 KB (65.535 bytes)

- Del 0000
- Al $FFFF = 2^16 - 1 = 65{,}535$
- Ejemplo: $A83F$

Direcciones en Memoria

Asigna números a cada *byte*

Ejemplo: memoria de 64 KB (65.535 bytes)

- Del 0000
- Al $FFFF = 2^16 - 1 = 65{,}535$
- Ejemplo: $A83F$

Ejemplo: memoria de 4 GB (4.294.967.296 bytes)

- Del 00000000
- Al $FFFFFFFF = 2^32 - 1 = 4{,}294{,}967{,}296$

Hexadecimales ofrecen una descripción compacta de direcciones en memoria

Direcciones en Memoria

Asigna números a cada *byte*

Ejemplo: memoria de 64 KB (65.535 bytes)

- Del 0000
- Al $FFFF = 2^16 - 1 = 65{,}535$
- Ejemplo: $A83F$

Ejemplo: memoria de 4 GB (4.294.967.296 bytes)

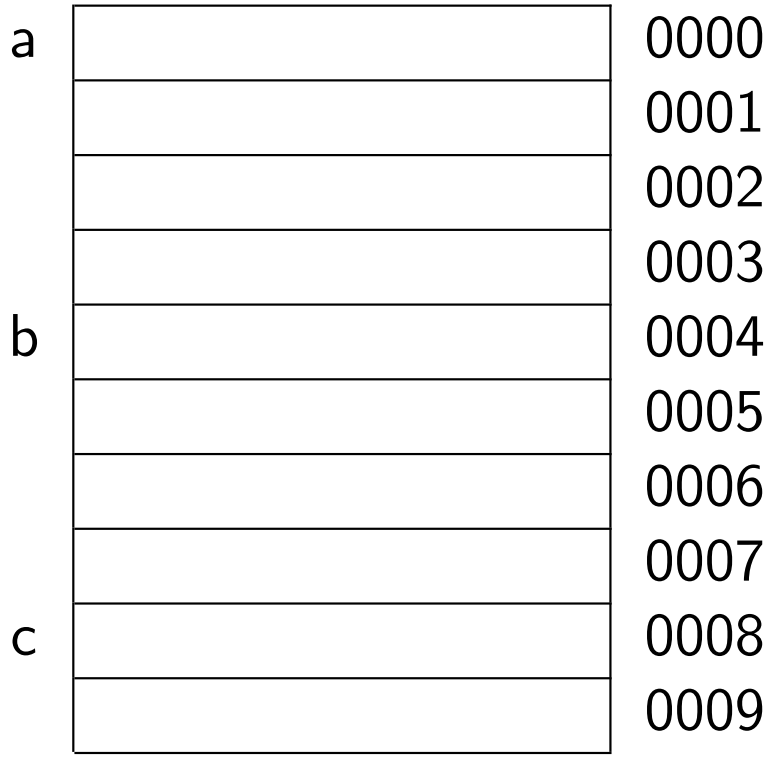
- Del 00000000
- Al $FFFFFFFF = 2^32 - 1 = 4{,}294{,}967{,}296$
- Ejemplo: $A83F19D2$

Hexadecimales ofrecen una descripción compacta de direcciones en memoria

Direcciones en Memoria - Ejemplos

Ejemplos con 64KB de memoria (16 bits para enumerar direcciones):

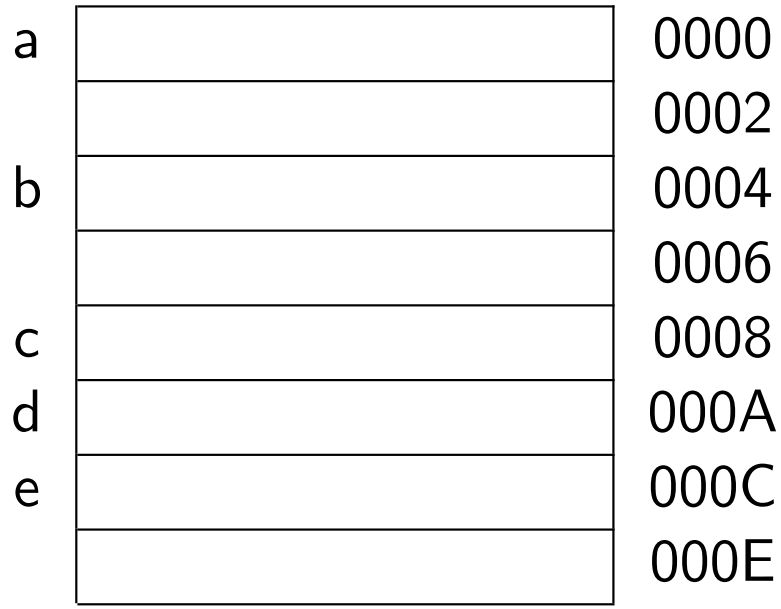
```
int a=10; int b=2; char c='K';
```



Direcciones en Memoria - Ejemplos

El mismo ejemplo en bloques de 16 bits:

```
int a=10; int b=2; char c='K'; char d = 'A'; int e = 5
```



Asignando memoria a variables

Asignación de memoria

Tres regiones de memoria:

Fija: constantes y variables estáticas. Se asigna memoria inicialmente.

Asignación de memoria

Tres regiones de memoria:

- **Fija**: constantes y variables estáticas. Se asigna memoria inicialmente.
- **Heap**: objetos creados dinámicamente (asignación dinámica de memoria)
  - Asignada al inicio (junto a la región fija)

Asignación de memoria

Tres regiones de memoria:

Fija: constantes y variables estáticas. Se asigna memoria inicialmente.

Heap: objetos creados dinámicamente (asignación dinámica de memoria)

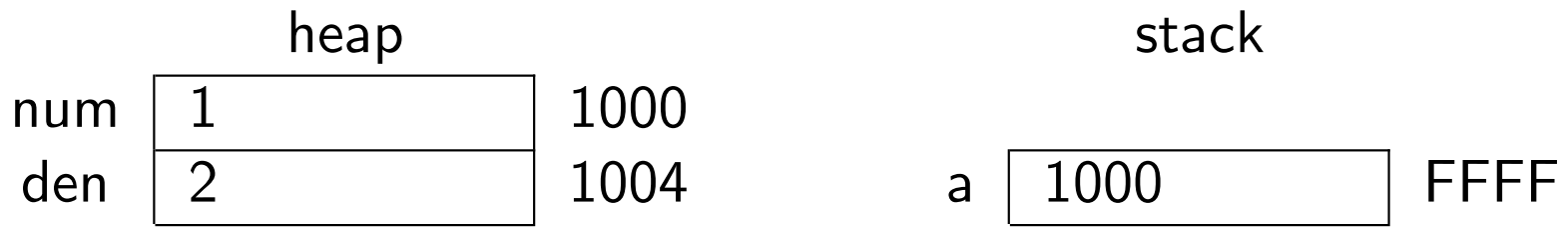
Asignada al inicio (junto a la región fija)

Stack: variables locales (definidas en los métodos)

Asignada el final (de la memoria asignada al programa)

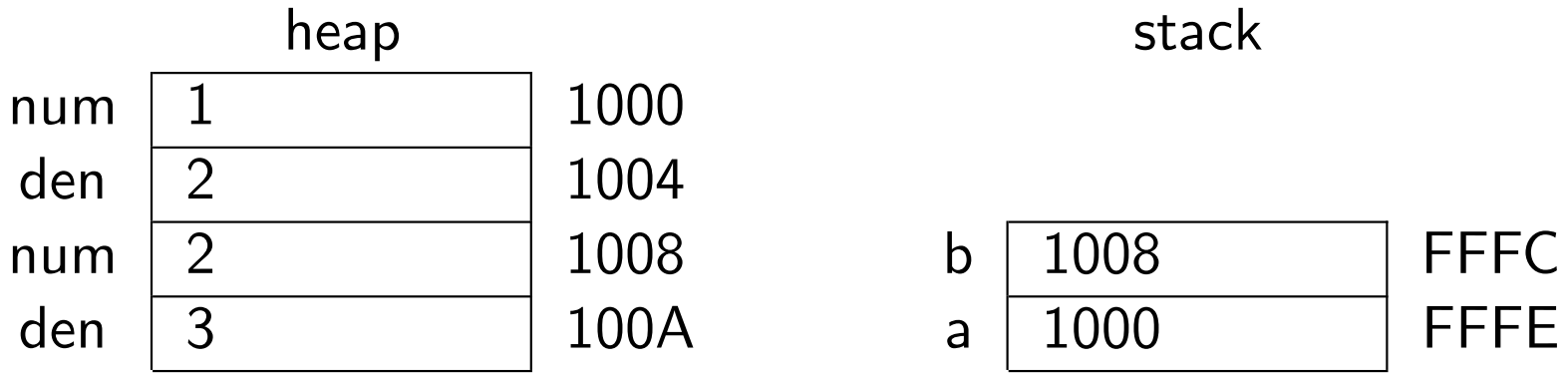
Asignación de memoria - Ejemplo

```
NumeroRacional a= new NumeroRacional(1, 2);
```



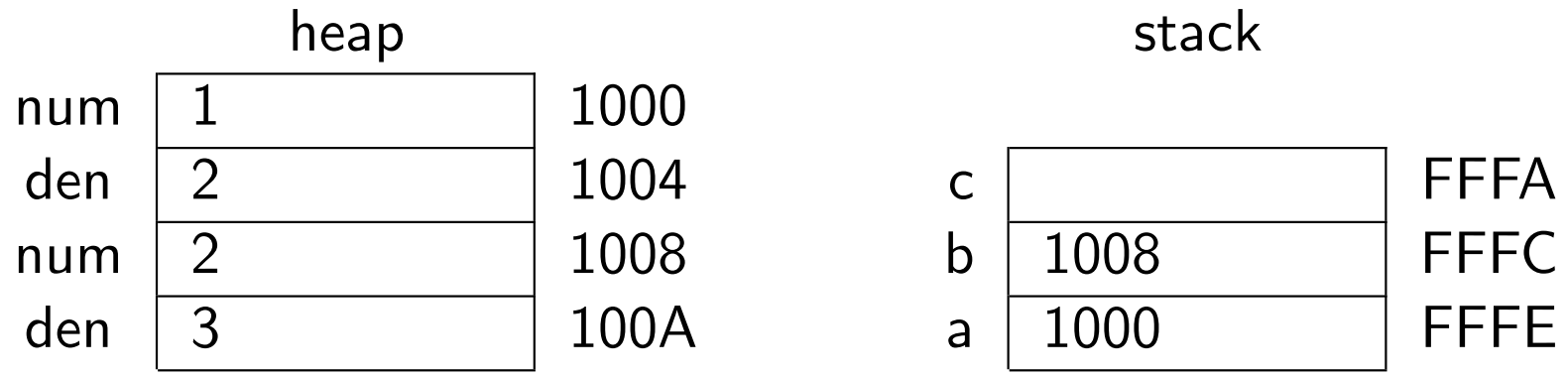
Asignación de memoria - Ejemplo

```
NumeroRacional a= new NumeroRacional(1, 2);
NumeroRacional b= new NumeroRacional(2, 3);
```



Asignación de memoria - Ejemplo

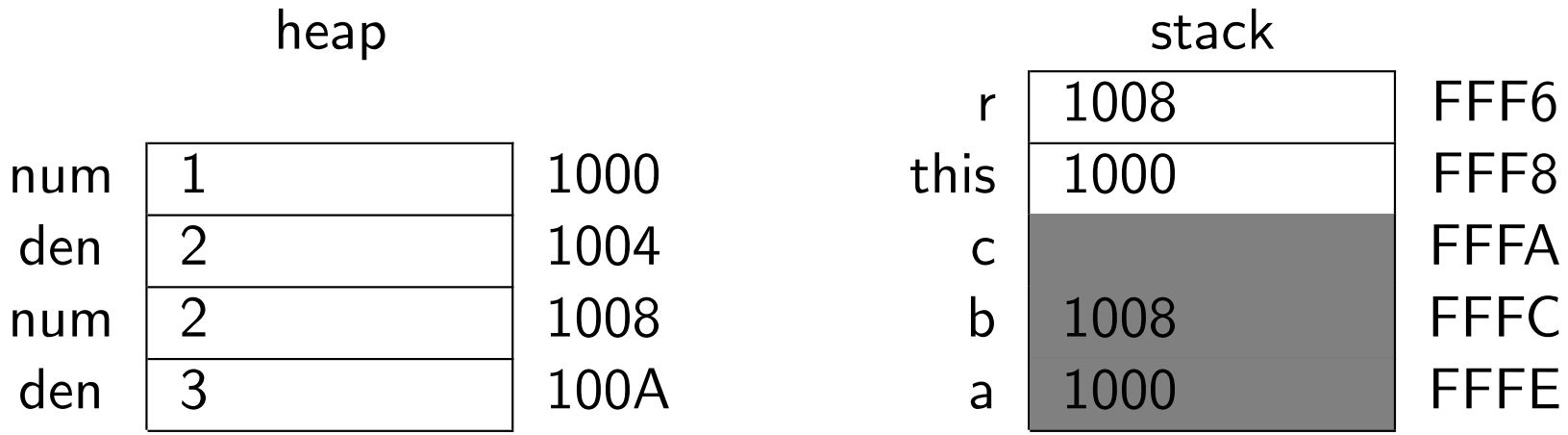
```
NumeroRacional a= new NumeroRacional(1, 2);
NumeroRacional b= new NumeroRacional(2, 3);
NumeroRacional c=a.suma(b);
```



Asignación de memoria - Ejemplo

```
NumeroRacional a= new NumeroRacional(1, 2);
NumeroRacional b= new NumeroRacional(2, 3);
NumeroRacional c=a.suma(b);
```

```
public NumeroRacional suma(NumeroRacional r){
 return new NumeroRacional(this.num*r.den + this.den*r.num,
     this.den*r.den);
}
```



Asignación de memoria - Ejemplo

```
public NumeroRacional suma(NumeroRacional r){
 return new NumeroRacional(this.num*r.den + this.den*r.num,
     this.den*r.den);
}
```

$$this.num * r.den + this.den * r.num = 1 * 3 + 2 * 2 = 7$$

$$this.den * r.den = 2 * 3 = 6$$

heap

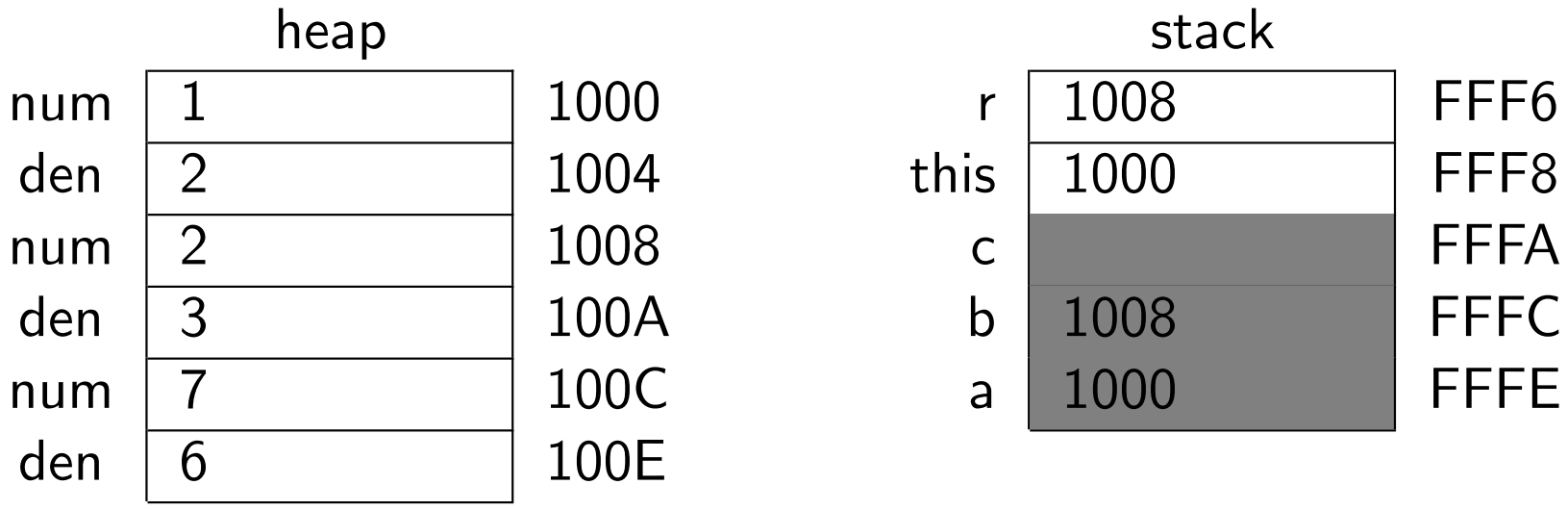
num	1	1000
den	2	1004
num	2	1008
den	3	100A

stack

r	1008	FFF6
this	1000	FFF8
c		FFFA
b	1008	FFFC
a	1000	FFFE

Asignación de memoria - Ejemplo

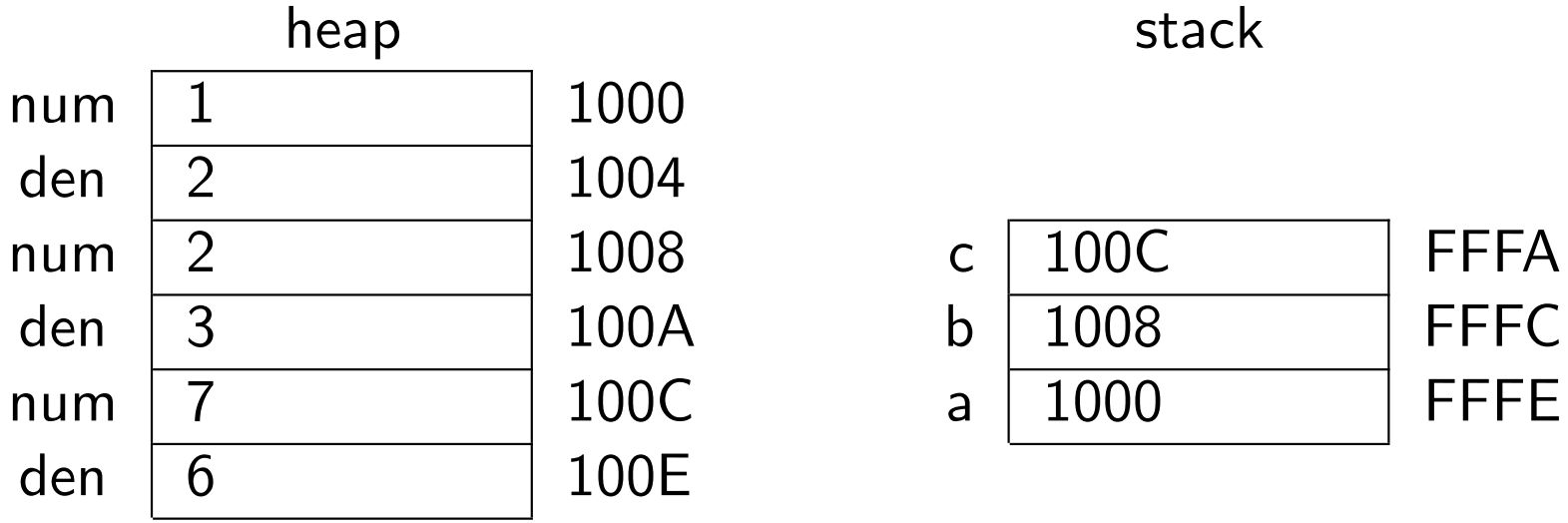
```
public NumeroRacional suma(NumeroRacional r){
 return new NumeroRacional(this.num*r.den + this.den*r.num,
     this.den*r.den);
}

 return new NumeroRacional(7, 6);
```



Asignación de memoria - Ejemplo

```
NumeroRacional a= new NumeroRacional(1, 2);
NumeroRacional b= new NumeroRacional(2, 3);
NumeroRacional c=a.suma(b);
```

```
public NumeroRacional suma(NumeroRacional r){
 return new NumeroRacional(this.num*r.den + this.den*r.num,
     this.den*r.den);
}
```



Asignación de memoria - Resumen

Manejo de memoria al usar objetos:

Asignación de memoria - Resumen

Manejo de memoria al usar objetos:

- Al declarar el objeto (variable local) reservamos un espacio en memoria (en el stack) donde mantenemos la dirección (en el heap) donde se almacena el objeto.

Asignación de memoria - Resumen

Manejo de memoria al usar objetos:

- Al declarar el objeto (variable local) reservamos un espacio en memoria (en el stack) donde mantenemos la dirección (en el heap) donde se almacena el objeto.
- Al llamar un método, se pasan las direcciones (en el heap) de los objetos al método. El método crea una copia de éstas direcciones (en el stack).

Asignación de memoria - Resumen

Manejo de memoria al usar objetos:

- Al declarar el objeto (variable local) reservamos un espacio en memoria (en el stack) donde mantenemos la dirección (en el heap) donde se almacena el objeto.
- Al llamar un método, se pasan las direcciones (en el heap) de los objetos al método. El método crea una copia de éstas direcciones (en el stack).
- Al retornar el método devuelve la dirección (en el heap) y esta se puede asignar a una variable local (en el stack).

Asignación de memoria - Cuidado

Como se pasan direcciones (**referencias**) de objetos al llamar métodos...

Asignación de memoria - Cuidado

Como se pasan direcciones (**referencias**) de objetos al llamar métodos...

Se puede alterar el valor de los objetos almacenados en estas direcciones.

Asignación de memoria - Cuidado

Como se pasan direcciones (**referencias**) de objetos al llamar métodos...

Se puede alterar el valor de los objetos almacenados en estas direcciones.

Al terminar la ejecución del método, la clase que lo llamó mantiene las mismas direcciones de los objetos, pero estos pueden haber cambiado de valor.

Asignación de memoria - Cuidado

Como se pasan direcciones (**referencias**) de objetos al llamar métodos...

Se puede alterar el valor de los objetos almacenados en estas direcciones.

Al terminar la ejecución del método, la clase que lo llamó mantiene las mismas direcciones de los objetos, pero estos pueden haber cambiado de valor.

Ejemplo: clases `Entero.java` y `AppletNumEntero.java` en el paquete `enteros` del proyecto `proyectoNumeros`

Objetos y tipos primitivos se manejan de manera diferente.

Clases Empaquetadoras (Wrappers)

Clases Empaquetadoras

Versiones en objetos de tipos primitivos

Permiten trabajar con tipos primitivos como objetos

Solo cuando sea necesario

Clases Empaquetadoras

Versiones en objetos de tipos primitivos

Permiten trabajar con tipos primitivos como objetos

Solo cuando sea necesario

Tipo primitivo	Clase Empaquetadora
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`boolean`	`Boolean`
`char`	`Character`

Ejemplos de Clases Empaquetadoras

```
public class AppletInteger extends ConsoleProgram{
 public void run(){
  Integer a = new Integer(10);
  Integer b = new Integer(10);

  println("a==b: "+(a==b));
  println("a.equals(b): "+(a.equals(b)));

  }
}
```

Ejemplos de Clases Empaquetadoras

```
public class AppletInteger extends ConsoleProgram{
 public void run(){
  Integer a = new Integer(10);
  Integer b = new Integer(10);

  println("a==b: "+(a==b));
  println("a.equals(b): "+(a.equals(b)));

  }
}
```

== compara si el objeto es el mismo (igual dirección en memoria)

equals() compara si los valores de los objetos son iguales

Ejemplos de Clases Empaquetadoras

```
public class AppletInteger extends ConsoleProgram{
 public void run(){
  Integer a = new Integer(10);
  Integer b = new Integer(10);

  println("a==b: "+(a==b));
  println("a.equals(b): "+(a.equals(b)));

  }
}
```

== compara si el objeto es el mismo (igual dirección en memoria)

`equals()` compara si los valores de los objetos son iguales

Funcionalidades de la clase `java.lang.Integer`
`https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html`