Arrays de objetos

Programación I - UNGS

Arrays

- Un **array** es una colección de valores con las siguientes propiedades:

Arrays

- Un **array** es una colección de valores con las siguientes propiedades:
  - Cada valor está identificado por un índice (comenzando por el 0)

Arrays

- Un **array** es una colección de valores con las siguientes propiedades:
  - Cada valor está identificado por un índice (comenzando por el 0)
  - Todos los valores son del mismo tipo

Arrays

- Un **array** es una colección de valores con las siguientes propiedades:
  - Cada valor está identificado por un índice (comenzando por el 0)
  - Todos los valores son del mismo tipo
  - El largo de la colección es fijo, y se determina durante su creación

Arrays de objetos

- Los valores de un array pueden ser de cualquier tipo: tipos primitivos, objetos, o incluso otros arrays (formando matrices)

Arrays de objetos

- Los valores de un array pueden ser de cualquier tipo: tipos primitivos, objetos, o incluso otros arrays (formando matrices)
- Al crear un array, todos los valores del array se inicializan al valor por defecto

Arrays de objetos

- Los valores de un array pueden ser de cualquier tipo: tipos primitivos, objetos, o incluso otros arrays (formando matrices)
- Al crear un array, todos los valores del array se inicializan al valor por defecto
- En el caso de un array de objetos, el valor por defecto es null!

Ejemplo

- Consideremos la siguiente clase

Ejemplo

- Consideremos la siguiente clase

Ejemplo

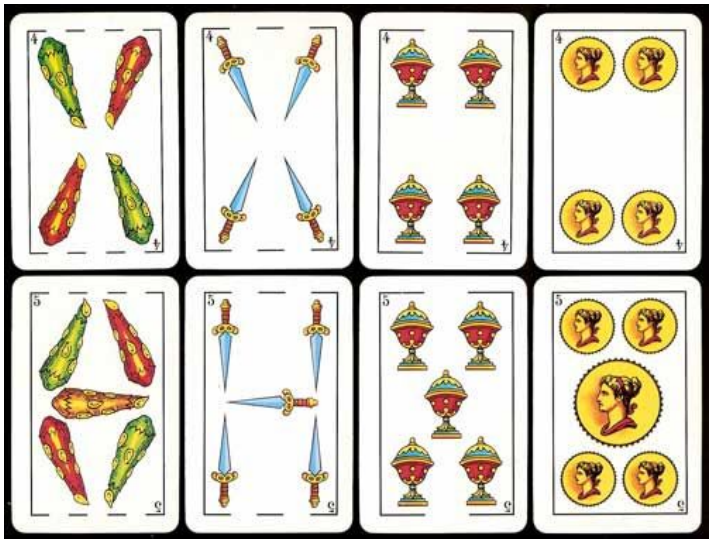
- Consideremos la siguiente clase

```
1  public class Carta {
2      int palo;
3      int numero;
4  }
```

Ejemplo

- Consideremos la siguiente clase

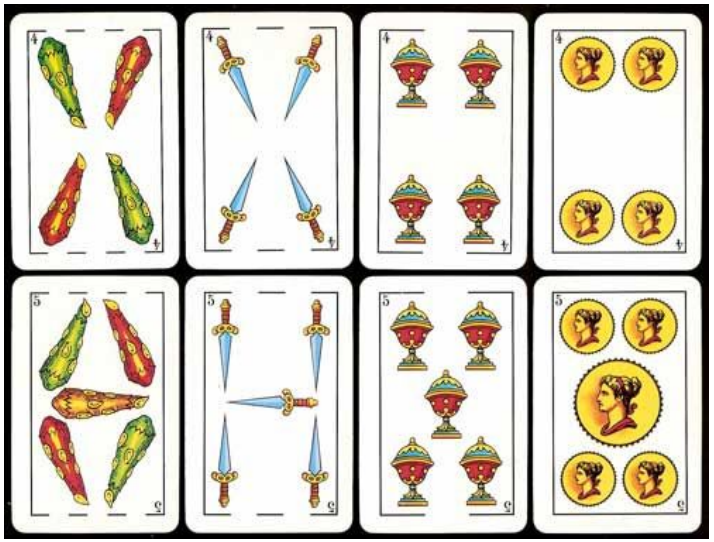
```
1  public class Carta {
2      int palo;
3      int numero;
4  }
```



Ejemplo

- Consideremos la siguiente clase

```
1  public class Carta {
2      int palo;
3      int numero;
4  }
```



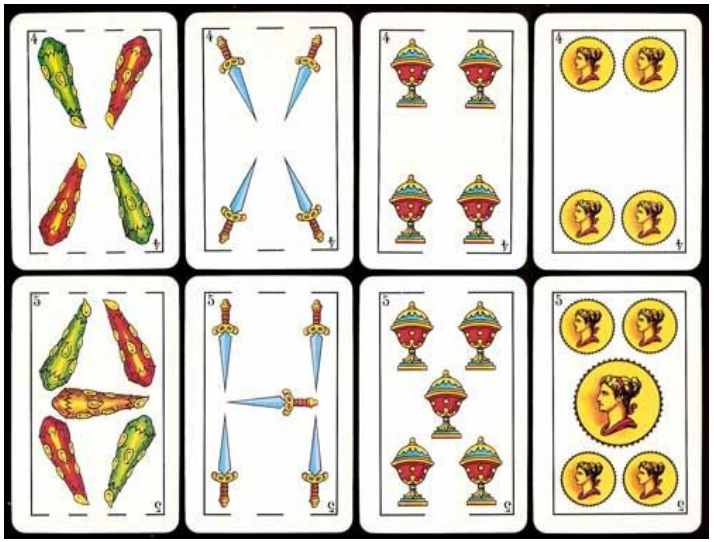
- ¿Qué resultado tendrá el siguiente código?

```
1  Carta[] cartas = new Carta[40];
2  System.out.println(cartas[0].numero);
```

Ejemplo

- Consideremos la siguiente clase

```
1  public class Carta {
2      int palo;
3      int numero;
4  }
```



- ¿Qué resultado tendrá el siguiente código?

```
1  Carta[] cartas = new Carta[40];
2  System.out.println(cartas[0].numero);
```

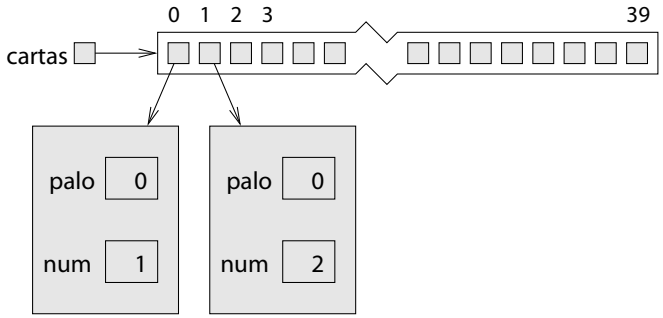
- NullPointerException: tratamos de acceder a un miembro (numero) de un objeto null (cartas[0]).

Inicialización de los objetos

- Por lo tanto, tenemos que inicalizar los valores del array.

Inicialización de los objetos

- Por lo tanto, tenemos que inicalizar los valores del array.
- El objetivo es que en cada posición del array tengamos una **referencia** a un objeto Carta.



Inicialización de los objetos

- Inicializamos el array con el siguiente código, que genera las cartas en orden y al mismo tiempo recorre en orden el array:

```
1   int j = 0;
2   for (int p = 0; p <= 3; p++) {
3       for(int i = 1; i <= 7; i++) {
4           cartas[j] = new Carta(p, i);
5           j++;
6       }
7       for (int i = 10; i <=12; i++) {
8           cartas[j] = new Carta(p, i);
9           j++;
10      }
11  }
```

Objetos que contienen arrays

- Ahora bien, podemos imaginar que este array de cartas es una variable de instancia de una clase Mazo:

```
1  public class Mazo {
2      Carta[] cartas;
3      public Mazo() {
4          // ¿Qué hace el constructor?
5      }
6  }
```

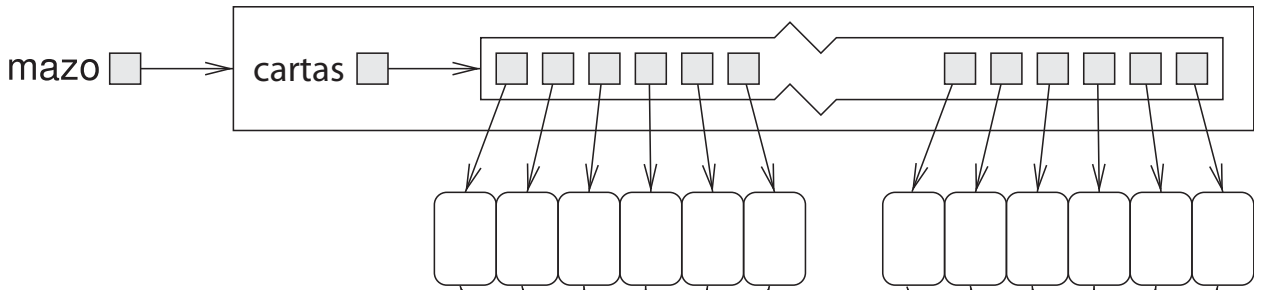
Objetos que contienen arrays

- Ahora bien, podemos imaginar que este array de cartas es una variable de instancia de una clase Mazo:

```
1  public class Mazo {
2      Carta[] cartas;
3      public Mazo() {
4          this.cartas = new Carta[40];
5          int j = 0;
6          for (int p = 0; p <= 3; p++) {
7              for (int i = 1; i <= 7; i++){
8                  cartas[j] = new Carta(p, i);
9                  j++;
10             }
11             for (int i = 10; i <= 12; i++){
12                 cartas[j] = new Carta(p, i);
13                 j++;
14             }
15         }
16     }
17 }
```

Objetos que contienen arrays

Entonces el código Mazo mazo = **new** Mazo() construye algo con la siguiente forma.



Aliasing

- Supongamos que la clase Mazo tiene el siguiente método de instancia

Aliasing

- Supongamos que la clase Mazo tiene el siguiente método de instancia

```
1  public Mazo copiar() {
2      Mazo mazo = new Mazo();
3
4      for(int i = 0; i < this.cartas.length; i++) {
5          mazo.cartas[i] = this.cartas[i];
6      }
7
8      return mazo;
9  }
```

Aliasing

- Observar que el método **Copiar** crea un nuevo Mazo, pero no crea nuevas cartas.

Aliasing

- Observar que el método **Copiar** crea un nuevo Mazo, pero no crea nuevas cartas.
- Las cartas del nuevo mazo son las mismas del mazo anterior.

Aliasing

- Observar que el método **Copiar** crea un nuevo Mazo, pero no crea nuevas cartas.
- Las cartas del nuevo mazo son las mismas del mazo anterior.

Aliasing

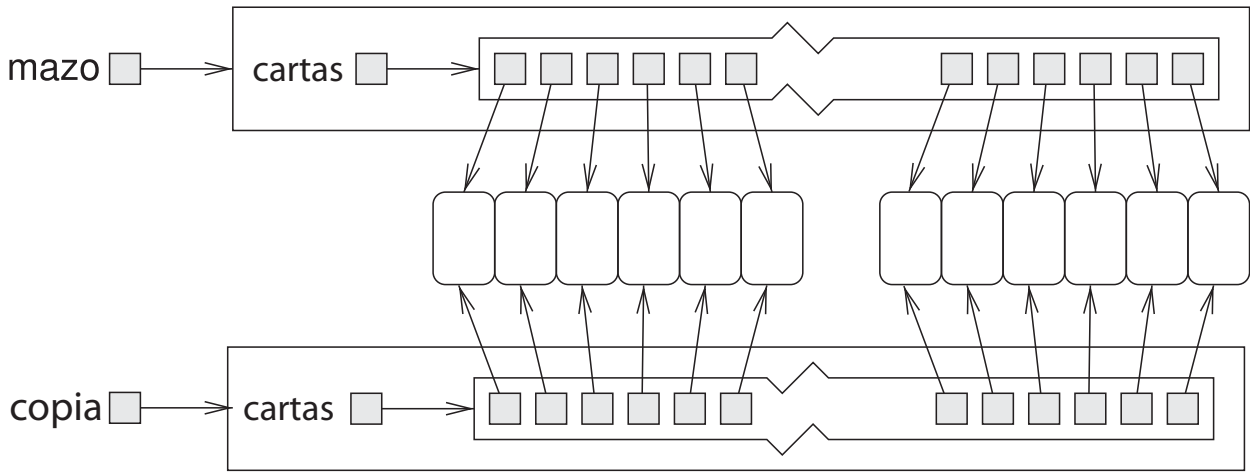
- Observar que el método **Copiar** crea un nuevo Mazo, pero no crea nuevas cartas.
- Las cartas del nuevo mazo son las mismas del mazo anterior.

```
1 Mazo mazo = new Mazo();
2 Mazo copia = mazo.copiar();
```

Aliasing

- Observar que el método **Copiar** crea un nuevo Mazo, pero no crea nuevas cartas.
- Las cartas del nuevo mazo son las mismas del mazo anterior.

```
1 Mazo mazo = new Mazo();
2 Mazo copia = mazo.copiar();
```



- Este tipo de copia se denomina **shallow copy**, o copia superficial.

Aliasing

- ¿Qué efecto tendría el siguiente código?

```
1 Mazo mazo = new Mazo();
2 Mazo copia = mazo.copiar();
3 copia.cartas[0].numero=2;
4 System.out.println(mazo.cartas[0].numero);
```

Aliasing

- La segunda opción para realizar la copia es que el nuevo Mazo tenga una copia de todas las cartas del Mazo original:

Aliasing

- La segunda opción para realizar la copia es que el nuevo Mazo tenga una copia de todas las cartas del Mazo original:

```
1  public Mazo copiar() {
2     Mazo mazo = new Mazo();
3     for (int i = 0; i < this.cartas.length; i++) {
4        if (this.cartas[i] == null) {
5           mazo.cartas[i] = null;
6        } else {
7           mazo.cartas[i] = new Carta(this.cartas[i].numero, this.cartas[i].palo);
8        }
9     }
10    return mazo;
11 }
```

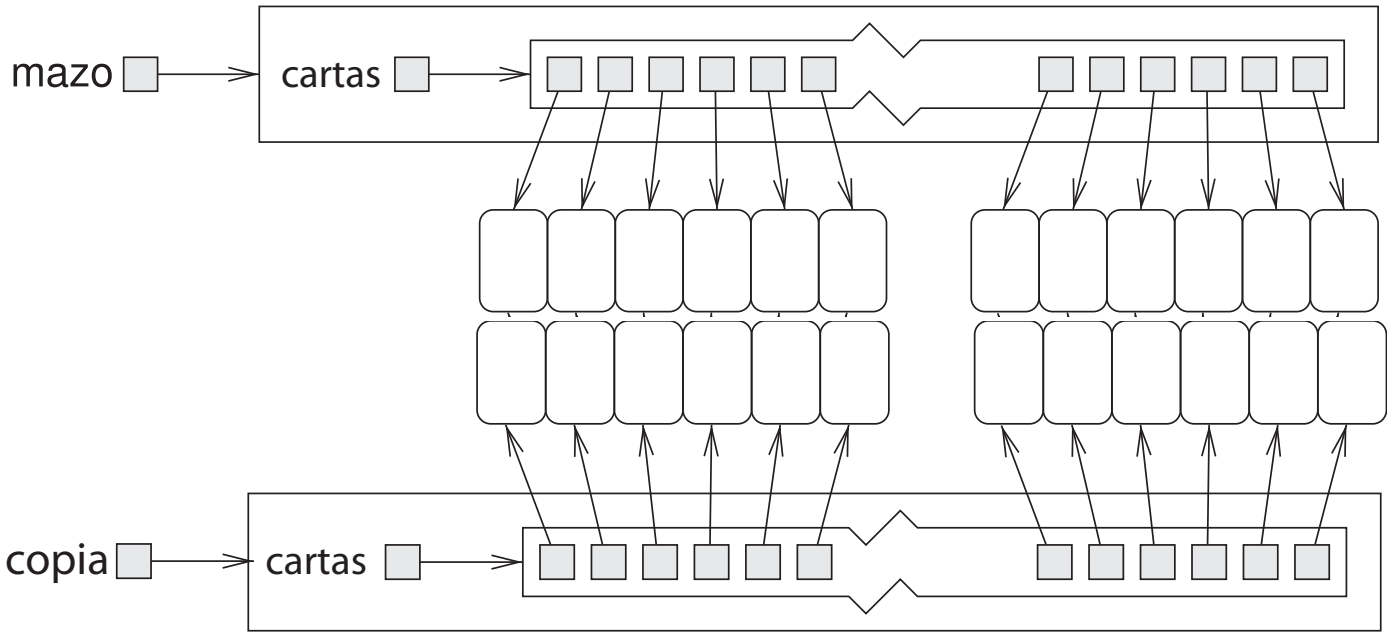
Aliasing

- La segunda opción para realizar la copia es que el nuevo Mazo tenga una copia de todas las cartas del Mazo original:
-

```
1  public Mazo copiar() {
2    Mazo mazo = new Mazo();
3    for (int i = 0; i < this.cartas.length; i++) {
4      if (this.cartas[i] == null) {
5        mazo.cartas[i] = null;
6      } else {
7        mazo.cartas[i] = new Carta(this.cartas[i].numero, this.cartas[i].palo);
8      }
9    }
10   return mazo;
11 }
```

- Este tipo de copia se denomina **deep copy**, o copia profunda.

Aliasing



Conclusión

- Los arrays de objetos, son una colección de variables de objeto (array[i]) como cualquier otra:

Conclusión

- Los arrays de objetos, son una colección de variables de objeto (array[i]) como cualquier otra:
  - Se inicializan en null

Conclusión

- Los arrays de objetos, son una colección de variables de objeto (array[i]) como cualquier otra:
  - Se inicializan en null
  - La asignación copia referencias

Conclusión

- Los arrays de objetos, son una colección de variables de objeto (array[i]) como cualquier otra:
  - Se inicializan en null
  - La asignación copia referencias
  - El operador == verifica si es el **mismo** objeto y no si sus campos son iguales

Conclusión

- Los arrays de objetos, son una colección de variables de objeto (array[i]) como cualquier otra:
  - Se inicializan en null
  - La asignación copia referencias
  - El operador == verifica si es el **mismo** objeto y no si sus campos son iguales
  - etc.

Conclusión

- Podemos componer expresiones. Por ejemplo:
  mazo.cartas[0].numero

Conclusión

- Podemos componer expresiones. Por ejemplo: mazo.cartas[0].numero
  - Al objeto mazo le pedimos la variable de instancia cartas(mazo.cartas). (Expresión del tipo $\mathrm{Carta}[]$)

Conclusión

- Podemos componer expresiones. Por ejemplo:
  mazo.cartas[0].numero
  - Al objeto mazo le pedimos la variable de instancia cartas(mazo.cartas). (Expresión del tipo $\mathrm{Carta}[]$)
  - A la variable de instancia cartas del objeto mazo, le pedimos aquella en la posición 0(mazo.cartas[0]). (Expresión del tipo Carta)

Conclusión

- Podemos componer expresiones. Por ejemplo:
  mazo.cartas[0].numero
  - Al objeto mazo le pedimos la variable de instancia cartas(mazo.cartas). (Expresión del tipo $\mathrm{Carta}[]$)
  - A la variable de instancia cartas del objeto mazo, le pedimos aquella en la posición 0(mazo.cartas[0]). (Expresión del tipo Carta)
  - A la carta en la posición 0 de la variable de instancia cartas del mazo, le pedimos el numero(mazo.cartas[0].numero). (Expresión del tipo int)

Conclusión

- Podemos componer expresiones. Por ejemplo:
  mazo.cartas[0].numero
  - Al objeto mazo le pedimos la variable de instancia cartas(mazo.cartas). (Expresión del tipo $\mathrm{Carta}[]$)
  - A la variable de instancia cartas del objeto mazo, le pedimos aquella en la posición 0(mazo.cartas[0]). (Expresión del tipo Carta)
  - A la carta en la posición 0 de la variable de instancia cartas del mazo, le pedimos el numero(mazo.cartas[0].numero). (Expresión del tipo int)
  - Así podríamos seguir componiendo, por ejemplo, pasándole esta expresión a cualquier función que admita un parámetro entero

En el libro...

Lo que vimos en esta clase, lo pueden encontrar en los **capítulos 11 y 12** del libro.