Guia de Repaso

Fundamentos de Java y OOP

Semana 01 - Lunes 09 de Febrero de 2026

Academia Java - Ciudad de Mexico

Temas del dia:

1. Clases, objetos y tipos primitivos
2. Mutabilidad: String vs StringBuilder
3. Variables static
4. Garbage Collector y referencias
5. Polimorfismo y herencia
6. Casting: upcasting, downcasting e instanceof

Instructor: Miguel Rugerio

1. Clases, Objetos y Tipos Primitivos

En Java existen dos grandes categorias de datos: los tipos primitivos (int, boolean, char, etc.) que se almacenan directamente en el stack, y los objetos que se crean en el heap usando el operador 'new'.

Proyecto: Inicio / v0

Se creo la clase Pato con un metodo volar() y una clase Principal que instancia el objeto y lo invoca. Este es el ejemplo mas basico de Programacion Orientada a Objetos.

Operador new

Reserva memoria en el heap y ejecuta el constructor. La variable guarda la REFERENCIA (direccion) al objeto, no el objeto en si.

NullPointerException

Si una referencia apunta a null y se intenta invocar un metodo sobre ella, Java lanza esta excepcion en tiempo de ejecucion.

```
Pato pato1 = new Pato();   // Se crea el objeto en el heap
pato1.volar();              // Imprime "Pato vuela"
// pato1 = null;
// pato1.volar();           // NullPointerException!
```

Primitivos vs Objetos

Caracteristica	Primitivos (int)	Objetos (Pato)
Almacenamiento	Stack	Heap
Contiene	El valor directo	Una referencia
Valor default	0 (si es campo)	null
Puede ser null?	No	Si

2. Mutabilidad: String vs StringBuilder

Un concepto fundamental en Java: algunos objetos son INMUTABLES (no se pueden modificar despues de creados) y otros son MUTABLES (si se pueden modificar).

Proyecto: Inicio / v1

Se demuestra que String.concat() NO modifica el String original (es inmutable), mientras que StringBuilder.append() SI modifica el objeto (es mutable). Los primitivos se pasan por valor, asi que las modificaciones locales no afectan al original.

```
int x = 10;
String cadena = "Hello";
StringBuilder sb = new StringBuilder("Hola");

show(x, cadena, sb);

System.out.println(x);       // 10     (primitivo: copia)
System.out.println(cadena);   // Hello  (inmutable: no cambio)
System.out.println(sb);       // Hola Mundo (mutable: SI cambio)
```

! String es inmutable: concat() crea un NUEVO String. Si no capturas el retorno, el resultado se pierde.

Proyecto: Inicio / v2 - Solucion correcta

La solucion es que el metodo RETORNE el nuevo String y el llamador lo reasigne:

```
cadena = show(x, cadena, sb);  // Se captura el retorno
// Dentro de show():
return cadena.concat(" World"); // Se retorna el nuevo String
```

Resumen de mutabilidad

Tipo	Mutable?	concat/append	Que pasa?
int (primitivo)	N/A	x += 10	Solo copia local
String	INMUTABLE	.concat()	Crea nuevo objeto
StringBuilder	MUTABLE	.append()	Modifica el mismo

3. Variables Static

Una variable static (de clase) pertenece a la CLASE, no a las instancias. Existe una sola copia compartida por todos los objetos.

Proyecto: statico / v0

La clase Pato tiene un campo 'static int contador' que se incrementa en cada constructor. Al crear 3 patos, el contador llega a 3 y las 3 instancias ven ese mismo valor.

```
public class Pato {
    String name;           // Variable de INSTANCIA (cada pato tiene la suya)
    static int contador;   // Variable de CLASE (compartida por todos)

    Pato(String name) {
        this.name = name;
        contador++;        // Incrementa el contador global
    }
}
```

```
new Pato("Donald");   // contador = 1
new Pato("Lucas");    // contador = 2
new Pato("Feo");      // contador = 3
Pato.contador;        // 3 (se accede con el nombre de la CLASE)
```

Instancia vs Static

Caracteristica	Variable de instancia	Variable static
Pertenece a	Cada objeto	La clase
Copias	Una por objeto	Una sola
Acceso	objeto.campo	Clase.campo
Ejemplo	pato1.name	Pato.contador

4. Garbage Collector y Referencias

El Garbage Collector (GC) de Java libera automaticamente la memoria de objetos que ya no tienen ninguna referencia activa. No necesitas liberar memoria manualmente.

Proyecto: gc / v0 - NewClass

Se demuestra paso a paso como las referencias cambian y cuando los objetos se vuelven elegibles para recoleccion:

```
Object obj = new Object();    // ObjA creado, ref: obj
NewClass tc = new NewClass();  // tc creado
tc.doSomething(obj);           // tc.o -> ObjA (2 refs)
obj = new Object();            // obj -> ObjB, ObjA aun vivo (tc.o)
obj = null;                    // ObjB sin refs -> elegible GC
tc.doSomething(null);          // tc.o -> null, ObjA sin refs -> elegible GC
```

Regla del GC

Un objeto es elegible para recoleccion cuando NINGUNA variable activa apunta a el, ni directa ni indirectamente a traves de otros objetos.

Proyecto: gc / v0 - TestClass (Nodos enlazados)

Un nodo n2 apunta a n1 via n2.next. Aunque anulemos la variable n1, el objeto sigue vivo porque n2.next lo referencia.

Proyecto: gc / v1 - Referencias circulares

Se crea un ciclo: n1 -> n3 -> n2 -> n1. La expresion (n1 == n3.next.next) es true porque recorrer la cadena nos regresa al mismo objeto. En Java, el GC PUEDE recolectar ciclos (a diferencia de lenguajes con reference counting).

> **!** El operador == compara REFERENCIAS (direcciones de memoria), no contenido. Para comparar contenido de objetos se usa .equals().

5. Polimorfismo y Herencia

Polimorfismo significa 'muchas formas': una referencia de tipo padre puede apuntar a cualquier objeto hijo, y el metodo que se ejecuta es el del OBJETO REAL, no el del tipo de la variable. Esto se conoce como dynamic dispatch.

Jerarquia de clases

```
Ave (padre)
     volar() -> "Volar Ave"
        |          |          |          |
      Pato      Pinguino    Perico    Murcielago
   "Volar     "No volar   "Volar     "Volar
    Pato"      Pinguino"   Perico"    Murcielago"
```

v0 - Polimorfismo basico

Una referencia tipo Ave puede apuntar a cualquier subclase:

```
Ave ave2 = new Pato();      // Upcasting automatico
ave2.volar();                // "Volar Pato" (se ejecuta el del objeto real)
```

v1 - Una sola variable polimorfica

Se reutiliza una sola referencia Ave reasignandola a diferentes subclases:

```
Ave ave;
ave = new Perico();    ave.volar();  // "Volar Perico"
ave = new Pinguino();  ave.volar();  // "No volar Pinguino"
ave = new Pato();      ave.volar();  // "Volar Pato"
```

v2 - Parametros polimorficos

Un metodo que recibe Ave funciona con CUALQUIER subclase:

```
static void comportamientoVolar(Ave ave) {
    ave.volar();  // Java decide en runtime cual version ejecutar
}
comportamientoVolar(new Pato());      // "Volar Pato"
comportamientoVolar(new Pinguino());   // "No volar Pinguino"
```

v3 - Arreglos polimorficos + Random

Un arreglo Ave[] puede contener mezcla de subclases. Se selecciona aleatoriamente:

```
Ave[] aves = {new Pato(), new Ave(), new Murcielago(),
              new Perico(), new Pinguino()};
int aleatorio = new Random().nextInt(aves.length);
return aves[aleatorio];  // Retorna cualquier subclase
```

6. Casting: Upcasting, Downcasting e instanceof

Casting es convertir una referencia de un tipo a otro dentro de la jerarquia de herencia.

Upcasting (automatico y seguro)

Asignar un objeto hijo a una referencia padre. Siempre es seguro y automatico:

```
Ave ave = new Pato();  // Upcasting: Pato -> Ave (automatico)
// Regla: SUPERCLASE = SUBCLASE  (PAPAS = HIJOS)
```

Downcasting (explicito y riesgoso)

Convertir una referencia padre a tipo hijo. Requiere cast explicito y puede fallar:

```
Ave ave = new Pato();
((Pato)ave).volarPato();  // OK: el objeto real SI es Pato

Ave ave2 = new Perico();
((Pato)ave2).volarPato(); // ClassCastException! No es Pato
```

v1 - instanceof (casting seguro)

Se verifica el tipo ANTES de hacer cast:

```
if (ave instanceof Pato)
    ((Pato)ave).volarPato();  // Solo si realmente es Pato
```

v2 - Pattern Matching (Java 14+)

Sintaxis moderna que combina verificacion + cast en una linea:

```
if (ave instanceof Pato patito)   // Verifica Y castea
    patito.volarPato();            // patito ya es tipo Pato
```

> ! v2A es un ANTI-PATRON: hacer cast sin instanceof causa ClassCastException cuando el objeto no es del tipo esperado. Siempre verificar primero.

v3 - Reglas de asignacion

Asignacion	Valido?	Razon
`B b = new B1()`	SI	Upcasting (auto)

`B1 b1 = (B1) b`	Depende	Downcast (puede fallar)
`B2 b2 = b1`	NO	Hermanas, no compatibles

7. Resumen del Dia

Proyecto	Tema principal	Concepto clave
Inicio v0	OOP basico	new, referencias, NPE
Inicio v1	Mutabilidad	String inmutable vs SB mutable
Inicio v2	Inmutables	Capturar valor de retorno
statico	Static	Variable compartida de clase
gc v0	Garbage Collector	Elegibilidad por referencias
gc v1	Refs circulares	GC maneja ciclos
polimorfismo v0-v3	Polimorfismo	Dynamic dispatch
cast v0-v3	Casting	instanceof y pattern matching

Estructura de los proyectos

Cada proyecto esta organizado en versiones (v0, v1, v2...) que evolucionan el mismo concepto progresivamente. Los archivos fuente estan en src/com/curso/vX/ y contienen comentarios explicativos en cada linea.

Para repasar

1. Clona el repositorio y abre los proyectos en Eclipse.
2. Ejecuta cada Principal.java version por version.
3. Lee los comentarios de cada linea para entender que hace.
4. Modifica el codigo y observa como cambia el comportamiento.
5. Intenta predecir la salida ANTES de ejecutar.