decodigo
.com

10 de las Mejores Prácticas en Java

abril 14, 2013 por ant

1.- Evitar la creación innecesaria de objetos, Lazy Initialitation

La creación de objetos en Java es una de las operaciones mas costosas en términos de uso de memoria e impacto en el performance. Esto es evitable creando o inicializando objetos solo en el momento en que serán requeridos en el código.

```
public class Paises {

    private List paises;

    public List getPaises() {
        //se inicializa solo cuando es requerido
        if(null == paises) {
            paises = new ArrayList();
        }
        return paises;
    }
}
```

2.- Nunca hacer variables de instancia públicas

Hacer una variable de instancia pública puede ocasionar problemas en un programa. Por ejemplo si tienes una clase MiCalendario. Esta clase contiene un arreglo de cadenas diasDeLaSemana. Pero es una arreglo público y este puede ser accedido por cualquiera. Tu asumes que este arreglo contiene siempre los 7 nombres de los días de la semana. Alguien por error puede cambiar el valor e insertar un error!

```
public class MiCalendario {

   public String[] diasDeLaSemana =
      {"Domingo", "Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado"};

   //mas código

}
```

La mejor práctica como mucho de ustedes saben, es definir siempre estas variables como privadas y crear los métodos accesores, “setters“ y “getters”

```
private String[] diasDeLaSemana =
   {"Domingo", "Lunes", "Martes", "Miercoles", "Jueves", "Sabado", "Domingo"};

public String[] getDiasDeLaSemana() {
   return diasDeLaSemana;
}
```

Pero escribir los métodos accesores no resuelve el problema del todo. El arreglo sigue siendo accesible. La mejor forma de hacerlo inmodificable es devolviendo un arreglo clonado en lugar del arreglo mismo. Esto se logra modificando el método get de la siguiente forma.

```
public String[] getDiasDeLaSemana() {
   return diasDeLaSemana.clone();
}
```

3.- Trata siempre de minimizar la Mutabilidad de las clases

Hacer una clase inmutable es hacerla inmodificable. La información de la clase se preservará durante el tiempo de vida de la clase. Las clases inmutables son simples y fáciles de manejar. Son “thread safe”. Normalmente son los bloques que permiten formar otros objetos más grandes.

No obstante, crear objetos inmutables pueden golpear significativamente el rendimiento de una aplicación. Así que elije cuidadosamente si quieres que una clase sea o no inmutable. Trata

siempre de tener clases pequeñas con el menor número de clases inmutables.

Para hacer una clase inmutable puedes definir sus constructor de forma privada y luego crear un método estático para inicializar al objeto y devolverlo.

```
public class Empleado {

   private String primerNombre;
   private String segundoNombre;

   // constructor private default
   private Empleado (String primerNombre, String segundoNombre) {
      this. primerNombre = primerNombre;
      this. segundoNombre = segundoNombre;
   }

   public static Empleado valueOf (String primerNombre, String segundoNombre) {
      return new Empleado (primerNombre, segundoNombre);
   }
}
```

4.- Trata de usar más las Interfaces sobre las Clases Abstractas

No es posible la herencia múltiple en Java, pero definitivamente puedes implementar múltiples interfaces. Esto hace que cambiar la implementación de una clase existente sea fácil y que puedas implementar una o mas interfaces en lugar de cambiar la jerarquía completa de la clase.

Pero si tu estás cien por ciento seguro de que métodos de una interface tendrás, entonces solo implementa esa interfaz. Es bastante difícil agregar un nuevo método en una interfaz existente sin alterar todo el código que se está implementando. Por el contrario un nuevo método puede ser fácilmente agregado en una clase abstracta sin alterar la funcionalidad existente.

5.- Limita siempre el alcance de una variable local

Las variables locales son grandiosas. Pero algunas veces pueden insertar mas bugs durante el copiado y pegado de código viejo. Minimizar el alcance de una variable local hace que el código sea mas legible, menos propenso a errores y mas mantenible.

Por lo tanto, debemos declarar variables justo antes de ser usadas.

Procura inicializar una variable desde su declaración. Si eso no es posible asígnale el valor nulo.

6- Trata de usar librerías estándar en lugar de hacer las tuyas desde cero

Escribir código es divertido. Pero no reinventes la rueda. Es bastante recomendable usar librerías estándar que ya han sido probadas, debugeadas y usadas por otros. Esto no sólo mejora la eficiencia de un programador sino que reduce las posibilidades de tener errores en el código. Además, usar una librería estándar hace al código mas legible y mantenible.

Por ejemplo Google tiene liberada la nueva librería Google Collections que puede ser usada para agregar mas funcionalidad a tu código.

7.- Siempre que sea posible trata de usar tipos primitivos en lugar de las clases Wrapper

Las clases Wrapper son buenas, pero también son lentas. Los tipos primitivos son como clases, sin embargo las clases wrapper almacenan la información completa acerca de una clase.

Algunas veces un programador puede agregar un error en el código usando una wrapper por un descuido. Por ejemplo:

```
int x = 10;
int y = 10;

Integer x1 = new Integer(10);
Integer y1 = new Integer(10);

System.out.println(x == y);
System.out.println(x1 == y1);
```

El primer System.out.println imprimirá true mientras que el segundo imprimirá false. El problema cuando comparas dos clases wrapper es que no se puede usar el operador ==, porque en realidad se están comparando referencias y no sus valores actuales.

Además si estás usando una clase wrapper no debes olvidar inicializarla. Porque el valor por default de las variables wrapper es null.

```
Boolean bandera = null;

      //más código

if(bandera == true) {
   System.out.println("Se establece el valor de bandera");
} else {
   System.out.println("No se establece el valor de bandera");
}
```

El código lanzará un NullPointerException cuando se trate de comparar con true y el valor sea nulo si en el código intermedio no fue inicializada.

8.- Usa los Strings con mucho cuidado

Usa siempre las cadenas con mucho cuidado en tu código. Una simple concatenación de cadenas puede reducir el rendimiento de tu programa. Por ejemplo si queremos concatenar cadenas usando el operador "+" en un ciclo for entonces todo el tiempo se estará creando un objeto String. Esto afecta tanto a la memoria como al rendimiento.

Además en lugar de que instancies una objeto String no uses su constructor, sino que debes instanciarlo directamente. Por ejemplo:

```
//instanciación lenta
String lenta = new String("solo otro objeto string");

//instanciación rápida
String rapida = "solo otro objeto string";
```

9.- Siempre regresa colecciones vacías en lugar de nulas

No importa que tu método regrese una colección o un arreglo, siempre asegúrate de que cuando sea necesario se regrese vacío y no nulo, en aquellos casos en los que no contendrá

elementos porque la lógica de tu programa lo requiera. Esto te ahorrará un montón de tiempo cuando hagas pruebas para valores nulos.

10.- El copiado defensivo es salvador

El copiado defensivo hace que los objetos creados estén libres de la mutación. Por ejemplo en el código siguiente tenemos definida la clase Estudiante la cual a su vez tiene una variable con la fecha de nacimiento que es inicializada cuando el objeto es construido.

```
public class Estudiante {
   private Date fechaNacimiento;

   public Estudiante(Date fechaNacimiento) {
      this. fechaNacimiento = fechaNacimiento;
   }

   public Date getFechaNacimiento () {
      return this.fechaNacimiento;
   }
}
```

Ahora podríamos tener el siguiente código que use al objeto Estudiante.

```
public static void main(String []arg) {
   Date fechaNacimiento = new Date();
   Estudiante estudiante = new Student(fechaNacimiento);
   fechaNacimiento.setYear(2019);
   System.out.println(estudiante.getFechaNacimiento ());
}
```

En el código siguiente creamos tan solo al objeto Estudiante con algunas fechas de nacimiento por default. Pero entonces cambiamos el valor del año de nacimiento. Después imprimimos el año de nacimiento, este año fue cambiado por 2019!

Para evitar estos casos, se puede utilizar el mecanismo defensivo copias. Cambie el constructor de la clase del estudiante a lo siguiente.

```
public Estudiante(Date fechaNacimiento) {
   this.fechaNacimiento = new Date(fechaNacimiento);
}
```

Esto para asegurarnos de tener otra copia de la fecha de nacimiento que usamos en clase Estudiante.

11.- Nunca dejes salir una excepción de un bloque finally

12.- Nunca lances «Exception» directamente.

tomado de: http://viralpatel.net

General

Java, mejores practicas

< Depuración

> Dale una Paleta

16 comentarios en «10 de las Mejores Prácticas en Java»

Anónimo

abril 15, 2013 a las 3:26 am

weno weno

Responder

Manuel López

mayo 30, 2013 a las 8:38 am

Muy bueno!

Os dejo un curso de Iniciación a Java online, por si estáis interedados.

Un saludo!

Responder

alejandro

marzo 14, 2019 a las 11:19 am

Hola me interesa

donde puedo hacerlo

ale_castillo8@yahoo.com.ar

Responder

Jorge Alberto Palma Díaz

octubre 14, 2013 a las 2:26 am

Muy bueno 😜 Solo una nota:

en el tip numero 10 si no me falla en el método Estudiante falta indicar que tipo de

datos recibira el método que para ese ejemplo es un objeto de tipo Date y creo que es asi

public Estudiante(Date fechaNacimiento) {

this. fechaNacimiento = fechaNacimiento;

}

si estoy mal un disculpa 🙄

Responder

Montero

noviembre 22, 2013 a las 10:08 pm

Es correcto le falta el tipo de dato 😀 Saludos

Responder

ant

junio 10, 2016 a las 4:36 pm

Gracias por la observación

Responder

elgogo52

diciembre 16, 2013 a las 1:00 am

Muy Buen aporte" Saludos desde Santo domingo RD 🙄

Responder

Kevin Mendez

junio 6, 2014 a las 8:05 pm

En el ejemplo 7 Declaras la variable bandera y abajo el if dice flag, deberia ser bandera tambien cierto?

Responder

Unknown

noviembre 25, 2015 a las 10:57 pm

Asi es

Responder

Anónimo

septiembre 26, 2014 a las 10:29 pm

Excelente !!!

Responder

Angelo Navarro

marzo 31, 2016 a las 5:38 pm

WENA WENA MONDINI

Responder

Javier Delgado Salcedo

abril 2, 2016 a las 12:59 am

baia baia :v

Responder

Alejandro Suasti

enero 18, 2017 a las 1:03 am

La solución del 10 no funciona

Responder

salomn

abril 19, 2017 a las 8:43 pm

me parece un buen post

Responder

llozano

junio 23, 2017 a las 5:16 pm

muy bueno gracias !!!!

Responder

Israel Orlando

abril 8, 2019 a las 6:10 pm

Gracias por los detalles

Responder

Deja un comentario

Nombre *

Correo electrónico *

Web

Guarda mi nombre, correo electrónico y web en este navegador para la próxima vez que comente.

Captcha

÷ one = 3

Publicar comentario

Herramientas Online

Crear código QR online

Ejemplos de Java

Crear archivo de texto

Leer una archivo de texto

Crear un archivo XML

Obtener información de un archivo

Leer archivo XML

Leer un archivo XML con SAX Parser

Crear un PDF con Apache FOP

Crear archivo Excel con Apache POI

Leer archivo Excel con Apache POI

Conexión a Base de Datos con JDBC

Extraer archivos de un ZIP

Crear archivos ZIP

Copiar archivo

Renombrar un archivo

Crear un directorio o carpeta

Listar y Filtrar Archivos con FileFilter

Borrar un archivo

Json con Google Gson

reCAPTCHAv2

FileUtils de Apache Commons-IO

Crear una Interfaz Gráfica con JavaFX

Visualizar Imagen en JavaFX

Codificación y Decodificación en Base64

Expresiones regulares

Ejemplos de Python

Crear un Web Server

Cliente de FTP

Crear archivo de texto

Leer un archivo de texto

Crear un archivo XML

Obtener información de un archivo

Leer archivo XML

Crear un PDF con ReportLab

Crear un archivo Excel

Leer un archivo Excel

Conexión a base de datos MariaDB

Extraer archivos de un ZIP

Crear archivo ZIP

Copiar archivo

Verificar si existen carpetas y archivos

Crear directorios o carpetas

Renombrar un archivo

Borrar un archivo

Crear Una Ventana o Interfaz Gráfica GUI

Crear una interfaz con PySimpleGUI

Expresiones Regulares

Detección de rostros con OpenCV

Mapeo de rostros con Dlib

Conexión a base de datos PostgreSQL

Ejemplos de Golang

Leer un archivo XML

Crear un archivo XML

Convertir un struct a JSON

Crear un Servidor Web

Hola Mundo

Leer archivo de texto

Crear archivo de texto

Conectar Golang con MariaDB

Ejemplos de C#

Crear archivo de texto

Leer archivo de texto

Crear archivo XML

Leer un archivo XML

Borrar un archivo

Crear, copiar y borrar archivos

Renombrar un archivo

Crear un directorio

Ejemplos de C++

Crear archivo de texto

Leer archivo de texto

Renombrar archivo