Módulo 1

Programación Java

Clase 1

- Repaso Java
- Estándares de codificación Java
- Buenas prácticas

Alejandro González Orellana

Temario

- **Repaso Java**
- Estándares de codificación Java
- Buenas prácticas

Temario

- Repaso Java
- Estándares de codificación Java
- Buenas prácticas

Estándares de codificación Java

Normas básicas

- Evitar archivos de más de 2000 líneas
- Estructura de un archivo:
  - Comentario inicial
  - Declaración de package
  - Declaración de imports
  - Doc-comment de la clase o interfaz
  - Implementación
- Simple utilizar packages
- Utilizar imports explícitos

Estándares de codificación Java

```
/*
 * CustomerData.java
 *
 * Last Modification
 * Author:   Agonzalez
 * Date:  08-jun-2005
 *
 * JaNoX Software Inc.
 */

package org.janox.customers.data;

import java.util.Map;
import java.util.HashMap

/**
 * Represents the basic data of a customer
 *
 * @author Agonzalez
 *
 */
public class CustomerData {
```

Comentario inicial, puede incluir datos de la última modificación, empresa, etc.

Siempre utilizar packages

Evitar imports del tipo `java.util.*`

Doc-comment, aparecerá en el Javadoc de la clase

Estándares de codificación Java

Indentación

Evitar líneas de más de 80 caracteres de largo, son incompatibles con algunas herramientas. Si una línea es muy larga dividirla

Se utilizan cuatro espacios como unidad de indentación

Estándares de codificación Java

Nomenclatura de nombres

- Nombres de paquete: Completamente en minúsculas, el nombre debe seguir el estandar. Ej: **com.janox.customers, com.janox.customers.data**
- Nombres de clase e interfaces: La primera letra de cada palabra debe ser mayúscula. Ej: **Item, CustomerData**
- Nombres de métodos y variables: La primera palabra en minúsculas, y el resto comenzando en mayúsculas. Ej: **addItem(), itemNumber, setName(), getName()**
- Nombres de constantes: Deben estar completamente en mayúsuculas, las palabras se separan con _ (*underscore*). Ej: **MAX_ITEM_NUMBER, DEFAULT_PREFIX**

Estándares de codificación Java

Declaración de variables

- Hacer una declaración por línea, permite poner comentarios
- Dar un valor inicial a la variable
- Siempre hacer declaraciones al inicio de un bloque
- Utilizar nombres significativos

Estándares de codificación Java

Sentencias

- Separar con un espacio la sentencia del paréntesis izquierdo
- Utilizar SIEMPRE llaves ({ y })
- Las sentencias vacías se escriben en una sola línea
- Si se desea ignorar una excepción indicarlo explícitamente

Estándares de codificación Java

```
for (int i = 0; i < MAX_VALUE; i++) {
    doSomething();
}
```

Utilizar llaves, aunque se trate de sentencias de una sola línea.

```
while (checkCondition());
```

Si la sentencia es vacía usar una sola línea

```
try {
    // do something...
} catch (Exception ignored) {}
```

El nombre de la excepción indica que no es un error la sentencia vacía

```
if (condition) {
    // do something...
} else if (condition 2) {
    // do something...
} else {
    // do something...
}
```

Los else están en la misma línea que la llave de cierre

Temario

- Repaso Java
- Estándares de codificación Java
- **Buenas prácticas**

Buenas prácticas

El nombre es importante

- **Error clásico:** Utilizar nombres que no indican nada sobre la variable o método
- **Solución:** Utilizar nombres descriptivos

```
// WRONG!!!
private String ln;

private int calculate(int amount) {
...
}

// RIGHT
private String lastName;

private int calculateTax(int amount) {
...
}
```

Buenas prácticas

El `null` es tu enemigo

- **Error clásico:** **`NullPointerException`** comparando objetos con literales
- **Solución:** Usar el literal primero

```
// UNSAFE!!!
if (myString.equals("")) {
   // do something...
}

// SAFE
if ("".equals(myString)) {
   // do something...
}
```

Buenas prácticas

El `null` es tu enemigo

- **Error clásico:** **`NullPointerException`** comparando objetos
- **Solución:** Utilizar comparaciones *nullsafe*

```
if (a != null && a.equals(b)) {
   // do something...
}

// Also true if both are null
if ( a != null ? a.equals(b) : b == null ) {
   // do something...
}
```

Buenas prácticas

El `null` es tu enemigo

- **Error clásico:** Se envían parámetros nulos a los métodos
- **Solución:** Realizar métodos *nullsafe* o simplemente lanzar excepciones

```
// Nullsafe
public boolean myMethod(String p_str) {
   boolean result = false;
   if (p_str != null) {
       // do something...
   }
   return result;
}

// With exceptions
public boolean myMethod(String p_str) {
   if (p_str == null) {
       throw new NullPointerException("p_str can not be null!");
   }
   // ...
}
```

Buenas prácticas

El `null` es tu enemigo

- **Error clásico:** Se envían colecciones o arreglos nulos a métodos
- **Solución:** Utilizar las facilidades de Java

```
// Empty collections

Collections.EMPTY_LIST
Collections.EMPTY_MAP
Collections.EMPTY_SET

// Zero size arrays
int[] myArray = new int[0];
final String[] EMPTY_STRING_ARRAY = new String[0];
```

Buenas prácticas

Ciclos

- **Error clásico:** Se evalua una condición en cada iteración de un ciclo
- **Solución:** Evaluar antes del ciclo

```
// WRONG!!!
for (int i = 0; i < myString.length(); i++) {
   // do something...
}

// RIGHT
int length = myString.length();

for (int i = 0; i < length; i++) {
   // do something...
}
```

Buenas prácticas

Ciclos

- **Error clásico:** Se concatenan `String` en un ciclo
- **Solución:** Usar `StringBuffer`

```
// WRONG!!!
String str = "";

for (int i = 0; i < MSG_NUMBER; i++) {
    str = str + messages[i];
}

// RIGHT
StringBuffer sb = new StringBuffer();

for (int i = 0; i < MSG_NUMBER; i++) {
    sb.append(messages[i]);
}

String str = sb.toString();
```

Buenas prácticas

Colecciones

- **Error clásico:** Amarrar una colección a una implementación específica
- **Solución:** Usar las interfaces de la API collections

```
// WRONG!!!
ArrayList list = new ArrayList();

// RIGHT
List list = new ArrayList();

// Sometimes better...
Collection itemCollection = new LinkedList();
```

Buenas prácticas

Colecciones

- **Error clásico:** Usar Vector o HashTable cuando no se necesita sincronización
- **Solución:** Usar las clases de la API collections

```
// Lists
List myList1 = new ArrayList();
List myList2 = new LinkedList();

// Maps
Map myMap1 = new HashMap();
Map myMap2 = new TreeMap();

// With synchronization
List synchronyzedList = Collections.synchronizedList(myList1);

// Unmodifiable
Map unmodifiableMap = Collections.unmodifiableMap(myMap1);
```

Buenas prácticas

“Magia”

- **Error clásico:** Utilizar números o strings “mágicos”
- **Solución:** Usar constantes con nombre

```
// WRONG!!!
for (int i = 0; i < 10; i++) {
    str = str + messages[i];
}

// RIGHT
for (int i = 0; i < MSG_NUMBER; i++) {
    str = str + messages[i];
}

public static final DEFAULT_PREFIX = “_”;
```

Buenas prácticas

Números

- **Error clásico:** Utilizar números de punto flotante para sumas monetarias
- **Solución:** Usar la clase `BigDecimal`

```
// WRONG!!!

public static final double TAX = 0.19;
double monto = 123.23;
...
result = monto * TAX;

// RIGHT
public static final BigDecimal TAX = new BigDecimal("0.19");
BigDecimal monto = new BigDecimal("123.23");
...
result = monto.multiply(TAX);
```