Curso de Programación en Java

Programación Orientada a Objetos

Java Code Conventions

- ¿Por qué tener convenciones?
  - Costo de mantenimiento
  - Mejoran la lectura de código
- Organización de los archivos
  - Las secciones deben estar separados por saltos de línea.
  - Comentarios opcionales para identificar las secciones
  - Evitar archivos de mas de 2000 líneas.

Java Code Conventions (cont.)

- Comentarios de inicio
  - Nombre de la clase
  - Información de la versión
  - Fecha
  - Copyright
- package
  - Primera línea de toda clase
  - Minúsculas

Java Code Conventions (cont.)

- Nombre de dominio de último nivel
  - com
  - edu
  - gov
  - mil
  - net
  - org
- Código ISO 3166

Java Code Conventions (cont.)

- import
  - Evitar incluir clases que no se utilizan
  - Evitar el uso de *
- Estructura de una clase
  - Comentarios de documentación
  - Sentencia clase o interface
  - Comentarios de implementación
  - Variables de clase
    - public
    - protected
    - package default
    - private

Java Code Conventions (cont.)

- Estructura de una clase
  - Variables de instancia
    - public
    - protected
    - package default
    - private
  - Constructores
  - Métodos
    - Agrupados por funcionalidad
    - Menos de 100 líneas

Java Code Conventions (cont.)

- Las líneas
  - 4 espacios como unidad de indentación.
  - Longitud
    - Evitar líneas de más de 80 líneas
  - Rompiendo las líneas
    - Despues de una coma
    - Antes de un operador

```
int variable = metodo(arg1,
                      metodo2(arg2,
                              arg3));
```

Java Code Conventions (cont.)

- Comentarios
  - De bloque
    - Descripción de archivos, métodos, estructuras de datos, algoritmos

```
/*
 * Comentario de bloque
 */
```

Java Code Conventions (cont.)

- Comentarios
  - De una línea
    - Precedidos de una línea en blanco

```
if (condicion) {

    /* Comentario de una linea*/
    . . .
}
```

Java Code Conventions (cont.)

- Comentarios
  - De fin de línea
    - No debe usarse para comentarios de múltiples líneas.
    - Delimitar fin bucles o sentencias de control

```
if (condicion) {

    /* Comentario de una linea*/
    . . .
} //Fin de condicion
```

Java Code Conventions (cont.)

- Declaraciones
  - Una declaración por línea
  - Un tipo por línea
  - Solo al principio de los bloques
- Clases e interfaces
  - Paréntesis de apertura inmediato al nombre de un método.
  - Llave de apertura en la misma línea
  - Llave de cierre en una línea nueva
    - Misma indentación que la línea de declaración
    - En la misma línea si no existen sentencias en el cuerpo

Java Code Conventions (cont.)

- Sentencias
  - if, if-else, if else-if
    - Usar siempre llaves para evitar errores

```
if (condicion) {

    /* Comentario de una linea*/
    . . .
} else {
    . . .
}//Fin de condicion
```

Java Code Conventions (cont.)

- Sentencias
  - for

```
for (int i = 0, i < 100; i++) {
    . . .
} //Fin de for
```

  - while

```
while (condicion) {
    . . .
} //Fin de while
```

Java Code Conventions (cont.)

- Sentencias
  - do-while

```
do {
    . . .
} while(condicion);
```

Java Code Conventions (cont.)

- Sentencias
  - switch

```
switch (condicion) {
case 0:
    . . .
    break;
default:
    . . .
    break;
} //Fin de switch
```

Java Code Conventions (cont.)

- Sentencias
  - try-catch

```
try {
    . . .
} catch (Exception ex) {
    . . .
}
```

Java Code Conventions (cont.)

- Sentencias
  - try-catch

```
/* Java 7 */
try {
    . . .
} catch (Exception | Exception2 ex) {
    . . .
}
```

Java Code Conventions (cont.)

- Nombres
  - Paquetes
    - Minúsculas
    - Nombre de dominio de alto nivel
    - ISO 3166
    - Subniveles de acuerdo a cada organización
      - Divisiones
      - Departamentos
      - Proyectos
      - Módulos

```
com.bstmexico
com.bstmexico.cursojava.dia2
mx.gob.sat
```

Java Code Conventions (cont.)

- Nombres
  - Clases e interfaces
    - Sustantivos
    - Camel Case
    - Simples
    - Descriptivos
    - Evitar acrónimos (HTML, RFC, URL)

```
public class Usuario
class ClienteServiceImpl
```

Java Code Conventions (cont.)

- Nombres
  - Métodos
    - Verbos
    - Camel Case, la primera letra siempre es minúscula
    - Infinitivo

```
public void ejecutar(){};
public void sumando(){}; //Evitar
public void calcularMonto(){};
```

Java Code Conventions (cont.)

- Nombres
  - Variables
    - Camel Case, la primera letra siempre es minúscula
    - Significativos
    - Evitar variables de in solo caracter
      - i, j, k, m, n índices temporales enteros
      - c, d, e caracteres

```
BigDecimal montoFactura;
byte edad;
int i;
```

Java Code Conventions (cont.)

- Nombres
  - Constantes
    - Mayusculas
    - Palabras separadas por “_”

```
static final int ALTURA_MAXIMA =
 999;
static final int ALTURA_MINIMA =
 100;
static final int CODIGO_ERROR = 0;
```

Práctica

Java Code Conventions

Tipos de dato primitivos

- byte
  - 8 bits
  - -128 a 127
  - Byte

```
byte a = 12;
byte b = 0;

Byte c = 32;
Byte d = -23;
```

Tipos de dato primitivos

- short
  - 18 bits
  - -32,768 a 32,767
  - Short

```
short a = -23;
short b = 2323;

Short c = 234;
Short d = 833
```

Tipos de dato primitivos (cont.)

- int
  - 32 bits
  - -2,147,483,648 a 2,147,483,647
  - Integer

```
int a = 0;
int b = 23423;

Integer c = 32;
Integer d = -2452;
```

Tipos de dato primitivos (cont.)

- long
  - 64 bits
  - -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
  - Long

```
long a = 232l;
long b = 972L;

Long c = 2342L;
Long d = -972423L;
```

Tipos de dato primitivos (cont.)

- float
  - IEEE 754 de 32 bits y precisión simple
  - Float

```
float f = 2343.23;
float g = -0.8530;

Float h = -2352.2839;
Float i = 0;
```

Tipos de dato primitivos (cont.)

- double
  - IEEE 754 de 32 bits y precisión simple
  - Double

```
double d = 23.57234d;
double e = 13423790092340.03;

Double f = -23312.34234;
Double g = 233.23423d;
```

Tipos de dato primitivos (cont.)

- boolean
  - Boolean

```
boolean b = false;
boolean c = true;

Boolean d = false;
Boolean e = true;
```

Tipos de dato primitivos (cont.)

- char
  - Unicode 16 bits
  - \u0000 a \uffff
  - Character

```
char a = ‘a’;
char b = ‘\u0043’;

Char c = ‘c’;
Char d = ‘\u0052’;
```

Práctica

Tipos primitivos

Casting entre tipos de dato

- Casting
  - Conversión de un tipo de dato primitivo a otro
  - Convertir un objeto de una clase a otra
    - Relación de herencia entre ambas
- Casting implícito
  - Tipos de datos compatibles
  - Tipo de dato destino es de rango mayor
  - No se necesita escribir código
  - Promoción

Casting entre tipos de dato

- Casting explícito
  - Información adicional al compilador.
  - Posible pérdida de información
  - Requiere escribir código

Práctica

Casting y Promoción

Operadores

- Aritméticos
  - + (adición, suma)
  - - (substracción, resta)
  - * (multiplicación)
  - / (división)
  - % (modulo)
  - ++ incremento
  - -- decremento

Operadores (cont.)

- Lógicos
  - > (mayor que)
  - < (menor que)
  - >= (menor o igual que)
  - <= (menor o igual que)
  - == (igual a)
  - != (distinto a)
  - && (and, “y” lógico)
  - || (or, “o” lógico)
  - ! (negación)

Operadores (cont.)

- Lógicos
  - ? :

Práctica

Uso de operadores

Programación Orientada a Objetos

- Paradigma para el desarrollo de software
- Basado en objetos
- Interacciones
- Beneficios
  - Desarrollo más rápido
  - Aplicaciones robustas y escalables
  - Código más entendible
  - Depuración mas fácil
  - Mayor soporte al cambio

Programación Orientada a Objetos (cont.)

- Objeto
  - Estructura de datos
  - Abstracción de un objeto del mundo real
  - Generados a partir de una clase
  - Estado (campos, atributos)
  - Comportamiento (métodos, funciones)
    - Alteran el estado interno
    - Mecanismo de comunicación con otros objetos

Programación Orientada a Objetos (cont.)

- Instancia
  - Cargar una clase en tiempo de ejecución
  - Operador “new”
    - Referencia a un nuevo objeto
    - Se usa en conjunto con un constructor
  - Declaracion
  - Construcción
  - Asignación

```
Triangulo triangulo;
triangulo = new Triangulo(3);

Cuadrado cuadrado = new Cuadrado(4);
```

Programación Orientada a Objetos (cont.)

- Constructores
  - Tipo específico de método
  - Mismo nombre que la clase
  - Crear e inicializar objeto
  - ¿Sin constructor?
  - Multiples constructores

Programación Orientada a Objetos (cont.)

- Modelado de objetos
  - Descripción de un conjunto objetos (estructuras) que forman parte del problema.
  - Interacciones

Práctica

Modelado de Software Orientado a Objetos

El banco XYZ requiere un control sobre los movimientos en las cuentas de los clientes. Por lo que necesita que los movimientos sean almacenados en una base de datos relacional o en otro archivo.

El detalle de los movimientos es enviado todos los días en un archivo de texto. El nombre del archivo tiene el siguiente formato:

yyyy_mm_dd_movimientos.txt

* donde:
    yyyy es el año representado en 4 dígitos.
    mm es el mes representado en 2 dígitos.
    dd es el día representado en dos dígitos.

Ejemplo: 2014_01_01_movimientos.txt

El archivo debe ser procesado y guardar la información en una base de datos relacional o en otro archivo dependiendo del resultado del procesamiento. Después de procesar el archivo de movimientos éste debe ser renombrado de la siguiente forma:

caso 1: Si el archivo fue procesado exitosamente (sin un solo error) debe agregarse .t al final del nombre del archivo, p.e.:

2008_01_01_movimientos.txt.t

caso 2: Si al procesar el archivo ocurre un solo error debe agregarse .err al final del nombre del archivo, p.e.:

2008_01_01_movimientos.txt.err

El contenido del archivo tiene el siguiente formato:

fecha|id establecimiento|id cliente|numero cuenta|monto

donde el formato de cada columna es:

1. fecha: yyyy-mm-dd hh:mm:ss
2. id establecimiento: un número entero largo positivo (8 bytes)
3. id cliente: un número entero largo positivo (8 bytes)
4. numero cuenta: cadena de 19 caracteres (ej: 0000-0000-0000-0000)
5. monto: un número con punto decimal (8 bytes) (ej: 1234.34)

Programación Orientada a Objetos (cont.)

- Clase
  - Plantilla de objetos
  - Plano para la construcción de objetos
  - Abstracción de un objeto del mundo real
    - Propiedades
    - Métodos
  - Unidad fundamental de programación en Java

Programación Orientada a Objetos (cont.)

- Clase
  - Estructura
    - Paquete
    - Imports
    - Cabecera
    - Campos o atributos
    - Métodos

Programación Orientada a Objetos (cont.)

- Variables de instancia
  - Cada instancia tiene su propia copia
  - Los tipos primitivos inician con un valor por default.
  - Se necesita una instancia
- Variables de clase (estáticas)
  - El valor es compartido por todas las instancias.
  - No se necesita una instancia

Práctica

Variables de instancia y de clase

Programación Orientada a Objetos (cont.)

- Métodos de instancia
  - Se necesita una instancia
  - Pueden operar sobre variables estáticas
  - Pueden invocar métodos estáticos
- Métodos de clase (estáticos)
  - Deben llevar la variable static
  - No se necesita crear un objeto
  - Operaciones comunes
  - Utilerias
  - No afectan las variables de instancia

Programación Orientada a Objetos (cont.)

- Métodos de clase (estáticos)
  - Contras
    - Se pierde flexibilidad
    - No se hace uso efectivo de memoria
    - Se contrapone a los principios de la POO

Práctica

Métodos de instancia y de clase

Características de la POO

- Paso por valor
  - Se hace una copia del valor original
  - Se modifica la copia
  - El valor original permanece intacto
- Paso por referencia
  - Se recibe la dirección en memoria del valor original
  - Si la función modifica el valor original

Características de la POO (cont.)

Java siempre pasa los parámetros por valor. Esta confusión se da debido a que todas las variables de objeto son referencias a objetos.

En el libro “The Java Programming Language” (Ken Arnold y James Gosling) “Existe un solo modo de paso de parametros en Java -paso por valor- y eso ayuda a mantener las cosas simples”

Práctica

Paso por valor y por referencia

Programación Orientada a Objetos (cont.)

- Clases abstractas
  - Declara métodos pero no su implementación
  - No se pueden instanciar
  - Clases hijas deben implementar o ser declaradas abstractas

Práctica

Clases abstractas

Programación Orientada a Objetos (cont.)

- Clases Internas
  - Permiten agrupar clases relacionadas
- Clases Internas Anónimas
  - Clases sin nombre
  - En tiempo de ejecución

Práctica

Clases Internas, clases internas anónimas

Programación Orientada a Objetos (cont.)

- Interfaces
  - Métodos abstractos y propiedades
- Ventajas
  - Desacoplamiento de código

Práctica

Interfaces