Temario Java

Understanding the Java Class Structure

Son el principal elemento del lenguaje, cuando se creá una clase se especificá las partes y características. Si se usan otras clases internas, se declara el objeto y en tiempo de ejecución crea instancía y/o espacio en memoria. Todos los diferentes objetos representan el estado del programa

FIELDS AND METHODS

Java tiene dos elementos primarios: métodos tambíen llamados funciones o procedures, en ciertos lenguajes, el segundo elemento son los campos(fields) comunmente conocidos como variables, las cuales representan es estado de un programa. Los métodos producen este cambio de estado en las variables(fields). El desarrollador crea la estrucutra y comportamiento de estos elementos(En capítulo 5 se ve diseño de clases.) .

Los metodos se les pueden pasar PARAMETROS para cambiar el estado de las variables method(Parameter or Parameters)

COMENTARIOS

Existen dos tipos de comentarios en java: Simples y multiples.

//Comentario simple representado por 2 diagonales, el compilador ignora esta línea. Generalmene se usa para indicar que termina alguna expresión

/\*

\* Comentario multiple respresentado por “ \* ” y “ / ” es comunmeten encontrado para documentacion \* de metodos y clases(java doc) la documentacion de java tiene una esctructura para una mejor

\* lectura.

\*/

CLASES VS ARCHIVOS

La mayoria de las veces las clases java están definidas dentro de su propio archivo (ClassName .java) Generalmente y por default son publicas nivel paquete por lo cual es posible invocarlas desde cualquier parte solo en el mismo paquete.

Ejemplo de default publico(package) al crearla:

class Animal{

String name=” La clase por dafault al crearla es publica(package), no se necesita espeficificar.”;

}

En un archivo se pueden crear 2 ó más clases pero solo una de ellas(1er clase declarada en el archivo) puede tener explicitamente la palabra public en caso contrario no compila.

MÉTODO MAIN()

Es el método principal de una clase java, en este metodo se ejecutan los procesos o métodos java a través de la JVM, La jvm a traves del sistema operativo solicita memoria al sistema operativo para asignar memoria para la ejecucion de los procesos, acceso a archivos entre otras cosas.

El tiempo de vida del metodo main dura mientras ejecuta todas las tareas que se le asignaron, lectura de archivos, ejecucion de varios métodos, etc.

Cuando se invoca la clase se pasan los argumentos que se especificaron en en metodo main(elementos del tamaño de la lista)

Ejemplo: java Zoo "San Diego" Zoo

Nota: Los argumentos que se le pasan al método pueden ser simple String o un arreglo donde los [] pueden estar de cualquier lado ya sea en el tipo de datos o en alias

**public** **class** Zoo {

**public** **static** **void** main(String[] args) { //String args[]

System.***out***.println(args[0]);

System.***out***.println(args[1]);

//AnyClass.getMethod(); ejecución de un método ubicado en cualquier clase.

}

}

UNDERSTANDING PACKAGE DECLARATIONS AND IMPORTS

Como en la vida real para localizar una dirección especifica de un domicilio, el cual corresponde a una persona en particular “Familia X”, de forma similar Java utiliza los “paquetes” dónde se específica la hubicación del las clases que se necesita invocar para poderlas utilizar dentro del desarrollo, esto debido a la existencia de clases con el mismo nombre pero de diferente procedencia.

Ejemplo

Date y Date

A simple vista es la misma clase sin embargo su naturaleza proviene de diferentes hubicaciones o paquetes y su uso es diferente.

La clase Date representa un tiempo determinado del tiempo(fecha), con mílisegundos de precisión. (fecha del sistema): **java.util.Date**

Date Representa valores de fecha en mílisegundos que permite al JDBC identificar el date con valor valido de sql: **java.sql.Date**

**Imports**

Para utilizar los paquetes en java se debe poner la palabra reservada “import” la cual el compilador de java lo utiliza para localizar que clase contiene la lógica de jecución, si se específica un paquete no valo, este puede no compilar y dar una descripción de la causa.

**WILCARDS**

Pueden ser utilizadas varias clases contenidas dentro del mismo paquete, en este caso se pueden utilizar los (\*) wildcard, este comidin se útiliza al final del path en el import, el compilador determina que puede estar dentro de esa ruta.

Ejemplo :

import java.util.Random

Cabe mencionar que esto no hace lento la ejecución del programa, de la misma forma no es aconsejable usar mucho los wildcards ya que para lectura de un nuevo desarrollador que retome un programa es legible si se especifica el paquete completo de donde proviene la clase, ya que como mencionamos hay clases con el mismo nombre pero con diferente uso, según su origen.

Hay que mencionar que hay imports “static” , se mencionara más a delante, solo basta con saber que existe de momento.

**RENDUNDANT IMPORTS**

Hay paquetes que el compilador tiene por default como java.lang, este se puede declarar pero no es necesario, el compilador lo interpreta, el uso de imports redundantes no evita que no compile el proyecto pero es una mala practica.

Ejemplo:

1: **import** java.lang.System;

2: **import** java.lang.\*;

3: **import** java.util.Random;

4: **import** java.util.\*;

5: **public** **class** ImportExample {

6: **public** **static** **void** main(String[] args) {

7:

Random r = **new** Random();

8:

System.out.println(r.nextInt(10));

9: }

10: }

Se llama import redundante porque se repite el paquete, aunque cambian las clases a utilizar, en el código anterior se muestra en linea 4 es redundante porque la clase Random esta dentro del import de la linea 3.

**NAMING CONFLICTS**

EL nombre de las clases en java no son unicos, por eso la importancia de usar paquetes, ejemplo con mismo nombre : java.util.Date and java.sql.Date, aunque proporcionan comportamiento diferente y provienen de diferente paquete, se llaman de la misma forma.

Se recomienda utilizar algun IDE para apoyarse con el auto completado y errores de compilacíon.

Cuando es necesario uitilizar dos clases con el mismo nombre se deben seguir ciertas reglas para que el compilador no marque error y sepa de donde viene cada una de ellas.

Ejemplo:

Una opción es utilizar en el import tradicional especificando el paquete de una de las clases, para la segunda clase en la declaración se especifica el nombre de dónde proviene.

**import** java.util.Date;

**public** **class** Conflicts {

Date date;

java.sql.Date sqlDate;

}

Otra opción es especificar en ambas declaraciones el paquete de donde provienen.

**public** **class** Conflicts {

java.util.Date date;

java.sql.Date sqlDate;

}

**CREATING A NEW PACKAGE**

Es recomendable crear y usar paquetes pára una mejor lectura de código, así como tambíen es buena práctica, el paquete por defaul sólo debe ser utilizado para pruebas, jamas para un proyecto en forma, se debe seguir una nomenclatura para que dentro de lo posible se pueda re utilizar el código.

El compliador detecta si es un paquete por default, se recomienda usar algun IDE que ayude a detectar errores de compilación por error en la estructura de paquete, así como también cuando se tienen clases con nombre similar el IDE nos puede ayudar a determinar de que paquete podemos hacer el import.

Es imoportante usar java a linea de comandos para entender el funcionamiento de java, que es lo que internamente hace el IDE en background.

Ejemplo de creacion de paquetes a nivel conosola.

Mac/Linux Setup

Crear dos archivos:

/tmp/packagea/ClassA.java

/tmp/packageb/ClassB.java

Cambiara al directorio previamente creado: cd /tmp

Compilar clases: **javac packagea/ClassA.java packageb/ClassB.java** esto creara los objetos compilados (.class)

Con los comandos anteriores se habrán creado los archivos y paquetes correspondientes

Ejecutando el código: **java packageb.ClassB** Sin la extencion de archivo '.java'

**CLASS PATHS AND JARS**

Sirve para especificar la hubicación de otros archivos explicitamente 'Class Path', estos pueden ser archivos de texto, archivos \*.jar, etc;

Los archivos **JAR**  son principalmente con archivos java los cuales pueden tener cierta logica y ejecutar diversas tareas.

Ejemplo: la siguiete instrucción indica que se utilizará un archivo .jar dentro de la clase. Cabe mencionar quedepende el S.O. es el comando a utilizar.

Linux / MAC

**java -cp ".:/tmp/someOtherLocation:/tmp/myJar.jar" myPackage.MyClass**

Windows

**java -cp ".;C:\temp\someOtherLocation;c:\temp\myJar.jar" myPackage.MyClass**

Se puede hacer uso de los wilcard(\*) para este mismo caso, ejemplo: Agregará todos los jars al class path

**java -cp "C:\temp\directoryWithJars\\*" myPackage.MyClass**

**CODE FORMATTING ON THE EXAM**

**Muchas veces se omite código para ahorrar espacio en el examen, ver que no siempre comienza con la línea número 1 en los ejemplos, por lo que se debe de dar por hecho que es código funcional. Se debe de dar prioridad cuando se menciona una línea en particular.**

**Se debe observar bien el número de linas omitidas o no para saber si es un código correcto o no.**

Ejemplo: este código no compila porque le falta el import y comienza linea 1 con la cración de la clase.

1: **public** **class** LineNumbers {

2: **public** **void** method(ArrayList list) {

3: **if** (list.isEmpty()) { System.out.println("e");

4: } **else** { System.out.println("n");

5: } } }

Se debe dar por hecho cuando no se muestre el método main() que este existe, al igual que los imports omitidos (dependiendo en que linea comienza) .

**CREATIN OBJECTS**

Un objecto es una instancia de una clase, un objeto al crearlo tiene constructor default el cual no se necesita especificar ya que el compilador lo interpreta, campos, inicializadores de instancia los cuales tienen un orden de inicializar.

**CONSTRUCTORS**

Al crear la instancia de una clase, al colocar la palabra reservada 'new', ejemplo

Random r = new Random();

Primero se declara el tipo de objeto a crear, en este caso (Random) , posteriormente el nombre de la variable o alias(r). Esto en java es referencia al objeto. Poseteriormente se crea el objeto con la palabra reservada 'new' más el objeto 'Random'.

Como se puede observar 'Random()' es seguido de los parentesis, a esto se le llama constructor y es un tipo de método especial cuando se crea un nuevo objeto.

Los constructores tienen 2 puntos clave: el nombre del constructor debe ser igual al nombre de la clase, y no debe de regresar níngun tipo de dato. El proposito de un constructor es inicializar atributos, se puede agregar adicionalemente cualquier tipo de código. Una forma de inicializar un atributo es directamente aunque no sea declarado dentro del constructor.

Ejemplo:

**public** **class** Chicken {

**int** numEggs = 0;// initialize on line

String name;

**public** Chicken() {

name = "Duke";// initialize in constructor

} }

El constructor es creado por default al crear el objeto por lo que no es necesario colocarlo, sin embargo se puede agregar si es indispensable, esto depende de nuetras necesidades al crear objetos.

Nota: Este no es un contructor ya que al poner el tipo de dato void se convierte en un método ordinario.

**public** **void** Chick() { }

**READING AND WRITING OBJECT FIELDS**

Es posible leer y escribir directamente en las variables de instancia en la invocación.

Ejemplo:

**public** **class** Swan {

**int** numberEggs;// instance variable

**public** **static** **void** main(String[] args) {

Swan mother = **new** Swan();

mother.numberEggs = 1;

// set variable

System.***out***.println(mother.numberEggs); // read variable

} }

Nota: Como se puede observar la variable numberEggs al no tener la palabra static, es de instancia ya que se invoca partir del objeto y no directamente de la clase.

Es fácil asignar y leer los valores de la variable numberEggs, sin embargo se puede asignar los valores de forma directa al declarar la variable, como se muestra en el siguiente ejemplo:

String first = "Theodore";

String last = "Moose";

String full = first + last;//Se puede concatenar y asignar valor previamente creados

**INSTANCES INITIALIZER BLOCKS**

El código que está entre llaves ({ }) es llamado “bloque de código”, a veces son bloques de código dentro de los métodos el cúal es ejecutado cunado el método es invocado, en general donde se muestran las ({}) se llama o representa bloques de código. El código que esta fuera de los métodos son llamados “**instance initializer**”

**public** **static** **void** main(String[] args) {

{ System.***out***.println("Feathers"); }

}

{ System.***out***.println("Snowy"); }

En este código existen tres bloques de código y un “**instance initializer**” cabe mecionar par que sean de este tipo tiene que tener una ({) que abre y otra (}) que cierra, en caso contrario el compilador marcara error.

**ORDER OF INITIALIZATION**

Cúando se escribe código inicializa campos en multipĺes lugares, se tiene que realizar un seguimiento del orden de inicialización, algunas reglas son:

* Campos y bloques de inicialización de instancia “**instance initializer**” se ejecutan en el orden en el que aparecen en el archivo Java.
* El constructor se ejecuta despúes de todos los campos y bloques de inicialización de instancia “**instance initializer**” se han ejecutado.

Ejemplo:

**1: public** **class** Chick {

2:**private** String name = "Fluffy";

3: {System.***out***.println("setting field");}

4: **public** Chick() {

5: name = "Tiny";

6: System.***out***.println("setting constructor");

7: }

8: **public** **static** **void** main(String[] args) {

9: Chick chick = **new** Chick();

10: System.***out***.println(chick.name); }}

**Salida en consola:**

setting field

setting constructor

Tiny //Al ultimo imprime esto ya que primero se ejecuta el constructor y luego se imprime name con Tiny

Como se observa lo que esta pasando aquí, Comienza con el método main() ya que es dónde java comienza con la ejecución. En la línea 9 se ejecuta el constructor de Chick.java. Java crea una nuevo objeto. Primero inicializa línea 2: name to "Fluffy". Despues ejecuta el print statement en el inicializador de instancia “instance initializer” en línea 3.

Una vez que todos los campos e inicializador de instancia “instance initializer” son ejecutados java regresa el constructor, la línea 5 cambia el valor de name a “Tiny”, línea 6 imprime, en este punto el constructor se esta ejecutando y regresa a ejecutar línea 10

**DISTINGUISHING BETWEEN OBJECT REFERENCES AND PRIMITIVES**

Java contiene dos tipos de datos: primitivos y tipos de referencia “reference type”.

Primitivos

Java tiene ocho tipos de datos incorporados. Esos ocho tipos de datos representan los bloques de construcción de objetos Java, porque todos los objetos de Java son sólo una colección compleja de estos tipos de datos primitivos.

Objetos tipo primitivos:



* Float y double son usados para números con punto flotante (decimales)
* Un float requiere poner la letra “f” seguido del punto flotante
* byte , short , int , y long son usados para números sin punto decimal.
* Cada tipo numérico utiliza el doble de bits de su antecesor, un byte ocupa 8 bits.

Ejemplo: short 8\*2 = 16. short usa el doble de bits.

uno se pregunta el valor maximo de los tipos de datos, sin embargo por ahora basta saber que un byte puede tener un valor de –128 to 127. un byte tiene 8 bits. Secuencia de valores usados para los diferentes tipos de datos: 2 × 2 = **4** × 2 = **8** × 2 = **16** × 2 = **32** × 2 = **64** × 2 = **128** × 2 = **256.**

El nùmero de bits usados por java para usar la memoria lo toma según el tipo de dato.

**Ejemplo:**  int num: Java reserva 32 bits para este caso, tomar la tabla mencionada anteriormente, así se asigna el valor en memoria.

Los tipos primitivos: Cuando un número es declarado en java, es llamado un literal. Por default, Java asume que está definiendo un valor int. Cuando una literal o tipo de dato primitivo rebasa el numero maximo permitido el compilador marcara error, pero si se le especifica en este caso L asume java que es tipo long recordemos que java por defaul lo toma como int.

*long max = 3123456789; // DOES NOT COMPILE;*

*long max = 3123456789L; // now Java knows it is a long*

Java le permite especificar los dígitos en varios otros formatos:

* octal (digits 0–7), which uses the number 0 as a prefix—for example, 017
* hexadecimal (digits 0–9 and letters A–F), which uses the number 0 followed by x or X

as a prefix—for example, 0xFF

* binary (digits 0–1), which uses the number 0 followed by b or B as a prefix—for exam-

ple, 0b10

Nota: Para el examen. Vas a tener que reconocer los valores literales válidos que se pueden asignar a los números.

En java 7 se agrega la funcionalidad de guiones (\_) en el uso de los literales, ejemplo:

**int** million1 = 1000000;

**int** million2 = 1\_000\_000;

La restricción que existe es cuando se usan decimales, una literal no puede terminar e inmediatamente poner '.' seguido de decimal o en su defecto depues de la literal poner cualquier cifra sin anteponer algun dígito E**jemplo**:

**double** notAtStart = \_1000.00;//No compila por no anteponer alguion un dígito

**double** notAtEnd = 1000.00\_;//No compila porque termina en guion

**double** notByDecimal = 1000\_.00;//No compila, despues del guion debe existir algun digito

**double** annoyingButLegal = 1\_00\_0.325;//Compila porque tiene dígito antes del punto decimal.

**REFERENCE TYPE**

Una referencia de tipo “reference type” refiere a un objeto(instancia de una clase). A diferencia de los tipo primitivos que mantienen sus valores in la memoria donde la variable es asignada, las referencias no guarda el valor, guarda una referencia "puntos" a un objeto de almacenamiento de memoria, dirección donde el objeto es asignado, un concepto de java no permite saber que dirección fisica de memoria es, sino solo usa la refencia del objeto.

* Un objeto puede ser asignado a otro objeto del mismo tipo.
* Una referencia puede ser asignado a un nuevo objeto con la palabra reservada “new ”.

String greeting = "How are you?";

java.util.Date today = **new** java.util.Date();

Ambos ejemplos son accedidos por su referencia en memoria.

**KEY DIFERENCES**

Existen algunas diferencias entre las referencias y los primitivos. Primero los de referencia pueden asignar valores nulos “null ”, el cual demuestra que no hay referencia de un objeto en memoria. Los tipos primitivos si se intenta asignar un “**null”** marcara errro de compilación.

Los tipos referencia pueden ser invocados de un metodo de referencia cuando no son nulos, los primitivos no tienen métodos declarados. **Ejemplo**:

String reference = "hello";

**int** len = reference.length();

**int** bad = len.length(); // DOES NOT COMPILE

Por ultimo los primitivos se declaran con minusculas, los de referencia comienzan con mayusculas, ya que son a final de cuentas una clase u objeto.

**DECLARING AND INITIALIZING VARIABLES**

Como se ha comentado una *variable es el nombre de una referencia en un almacenamiento de datos esta necesita tener un estado asignadole un nombre,* ***Ejemplo***

String zooName = "The Best Zoo";

**int** numberAnimals = 100;

En el ejemplo se muestra como inizializar una variable, primero se coloca el tipo o estado y posteriormente se asigna el valor correspondiente al estado de dicha variable.

**DECLARING MULTIPLE VARIABLES**

Se pueden declarar e inicializar multiples variables en la misma declaración, en el examen pueden venir trucos para confundir, **ejemplo**: int num, String value; // DOES NOT COMPILE.

Para que se código correcto se deben seguir las sigueintes reglas:

* Deben ser del mismo tipo las variables: int, String, etc. **boolean** b1, b2;
* Se puede solo declarar y/o inicializar en la misma declaración: String s1 = "1", s2;
* Java **no** permite declarar dos diferentes tipos en la misma declaración si se usa una (,) como separador: **double** d1, **double** d2;
* Java permite declarar dos diferentes tipos en la misma declaración si se usa una (;) **int** i1; **int** i2;
* No se permite usar (;) si no se especifica el tipo para la siguiente variable: **int** i3; i4;

En el examen se puede presentar varaibles declaradas en la misma línea para ahorrar código y tratar de confundir, en el mundo real no se recomienda esta practica.

**IDENTIFIERS**

Java tiene reglas sobre identificadores o palabras reservadas que usa el lenguaje, los cuales no podemos utilizar, es fácil identificarlos ya que el compilador nos marca error.

Las tres reglas de identiifcadores son:

* El nombre debe iniciar con una letra o el simbolo: $ ó \_
* Los siguientes caracteres también pueden ser números
* No puedes utilizarel mismo nombre que las palabras reservadas de Java.

**Algunas palabras reservadas son.**

Abstract, assert, boolean, break, byte, case, catch, char, class , const\*, continue, default, do, double, else, enum, extends, false, final, finally.

No legales:

3DPointClass // identifiers cannot begin with a number

hollywood@vine // @ is not a letter, digit, $ or \_

\*$coffee // \* is not a letter, digit, $ or \_

public // public is a reserved word

Aunque se pueden usar nombres locos sion sentido, es recomendable nombrarlas según su uso, algunas recomendaciones son:

* Métodos y variables, sus nombres ser con notación camello.
* Nombres de clases comienzan con mayuscula y es seguido de notación camello en caso de palabra compuesta, no usar el simbolo $ aunque sea correcto, el compilador lo usa para algunos archivos.

Java no usa solo elalfabeto ingles, soporta unicode, por lo que soporta más de 45,00 caractéres con las que se puede iniciar los identificadores de Java.

**Understanding Default Initialization of Variables**

Antes de utilizar una variable necesita un valor, algunas se inician en automatico y otras requieren ser inicializadas, vamos a ver las diferencias entre los valores predeterminados para los locales, instancia y

variables de clase.

**Local Variables**

Una variable local esta definida dentro de un método, este tipo de variables debe ser inicializada antes de usarla ya que no tiene un valor default, el compilador no permite leer un valor no inicializado

Ejemplo:

**public** **int** notValid() {

**int** y = 10;

**int** x;

**int** reply = x + y; // DOES NOT COMPILE because x isn't initialized

**return** reply;

}