**Tema 1: Breve introducción a Java**

* Java es un lenguaje de programación orientado a objetos.
* Se rige con la filosofía “Write one, run anywhere” (Es portable)
* Al compilar el código no se genera lenguaje máquina, sino un código intermedio conocido como Bytecode.
* La única máquina capaz de ejecutar Bytecode es la “Java Virtual Machine” (JVM).
* La JVM es especificada por Oracle y es implementada por varias empresas. Osea, existen distintas “marcas” de JVM (Oracle, IBM, …)
* Cada proceso Java se ejecuta en una JVM aparte.
* Java + JVM + Utilidades para programar, conforman una plataforma de software para el desarrollo de aplicaciones de usuario.

Existen varias plataformas en Java:

* SE (Standart Edition) Aplicaciones de escritorio.
* ME (Mobile Edition) Dispositivos móviles.
* EE (Enterprise Edition) Aplicaciones Servidor.

Java se distribuye de varias formas:

* JRE (Mínimo para ejecutar programas)
* JDK (JRE + SE)
* Servidores de aplicaciones (Implementaciones EE)

**Paradigmas de programación**

* En general, es una forma de ver/entender/modelar el mundo.
* En el ámbito del software, es un estilo fundamental de programación. Determina cómo el programa “ve” el mundo.
* Determina cómo debe ser usado el lenguaje por el programador.
* Altos lenguajes son multiparadigmas (no se hace una selección explicita del código).

Hay cuatro paradigmas principales:

Funcional

Lógico

Imperativo

Orientado a Objetos

Paradigma Lógico:

* El mundo se modela mediante predicados lógicos.
* Se aplican directamente principios matemáticos.
* Poco utilizado en aplicaciones comerciales.
* Prolog es el principal lenguaje.

Paradigma Funcional:

* El mundo se modela como funciones matemáticas.
* Lenguajes 100% funcionales (Lisp, Scheme).
* Permite declarar funciones, pasar funciones, pasar parámetros, retornar funciones, entre otros.
* Muchos lenguajes modernos han incorporado este paradigma por conveniencia.

Paradigma Imperativo:

* El mundo se modela como instrucciones, pasos, procedimientos.
* Ampliamente utilizados en aplicaciones comerciales.
* Tienden a ser muy eficientes (generalmente compilan en lenguaje máquina).
* Gramaticalmente simples.
* Se consideran como de nivel de abstracción bajos.
* Ejemplos: C/C++

Paradigma Orientado a Objetos:

* El mundo se modela como objetos del mundo real junto con su interacción.
* Muy naturales para los humanos.
* Facilitan la reutilización de código (Herencia).
* Mayoritariamente no generan lenguaje máquina.
* Poseen muchas estructuras sintácticas
* Ejemplos: Java, C++, Python, JavaScript.

**Lenguaje Java**

**Colección:** Matrices (puede contener datos primitivos “int, double, char” u objetos “Integer, Double, String”, siempre y cuando sean del mismo tipo y Listas acepta Objects (Cualquier tipo de elemento, no hay que definir el tamaño de la mismo).

**Conjunto:** Es una construcción de colecciones que por definición contiene elementos únicos, es decir ninguno duplicado. Un conjunto se preocupa por la singularidad, no por el orden de los datos como en el caso de las colecciones tipo matriz(array) y tipo List.

**Excepciones:**

* **Tiempo de compilación:** Errores de sintaxis (olvidar poner punto y como, cerrar llaves, etc)
* **Tiempo de Ejecución:** Heredan de la clase Throwable
* **Error:** Muy relacionado con problemas de hardware (escases de memoria, desbordamiento de la pila del sistema, etc. Es muy poco normal que ocurran.
* **Exception:**
* **IOException:** Clase que contiene excepciones verificadas (que un archivo no se encuentre, que se quiera leer y este vacié, entre otros). Requieren de try-catch
* **RuntimeExcepcion:** Clases que contiene excepciones no verificadas (Dividir entre cero, agregar más elementos a una matriz con magnitud definida, almacenar un dato en un tipo de dato incorrecto, etc).

**Nota:** Hay clases que exigen el lanzamiento de excepciones (throws), esto es implementar las instrucciones try-catch. Por ejemplo, la clase ImageIO exige captar una posible excepción con try-catch, en todo caso dicha excepción seria verificada ya que pertenece a la excepción IOException.

**Nota:** Las excepciones de tipo o que heredan de RuntimeExcepcion no exigen captar la excepción en un try-catch, es decir como programador te permite considerar código adicional que evite que tal excepción ocurra y de ser así hacer algo al respecto.

**throws y throw:** throws se utiliza en la declaración de un método para indicar que dicho método podría lanzar una excepción, mientras que la instrucción throw se utiliza en cualquier parte del código para indicar que en tal parte en concreto se lanza una excepción.

**¿Como crear excepciones?**

Se debe crear una clase que herede de la clase Exception o cualquier otra subclase de excepciones (RuntimeException, IOException, etc). La clase creada debe tener mínimo 2 métodos constructores, uno sin argumentos y el otro con un argumento tipo String para informar la causa de la excepcion. Hay que considerar que, si la clase hereda de la clase Exception o IOException, al llamar el método que podrá usar la excepción creada, el mismo deberá estar dentro de un try – catch, caso contrario si la clase hereda de la clase RuntimeException.

**Lectura y Escritura de archivos**

**FileWriter, BufferedWriter, PrintWriter:** Se usan normalmente para escribir datos orientados a caracteres en un archivo. A diferencia de la clase FileOutputStream, no necesita convertir la cadena en una matriz de bytes porque proporciona un método para escribir la cadena directamente.

**FileOutputStream:** Convierte un String en bytes y escribe los bytes en un archivo.

**RandomAccessFile:** Permite escribir en cualquier parte del archivo (parecido a Python).

**FileChannel:** Para escribir archivos muy grandes.

UML: Lenguaje de modelo unificado. Conjunto de diagramas. Lectura y ponerlo en la bitácora.

**Programación Orientada a Objetos**

**Paquetes:** Mecanismo para encapsular clases, sub paquetes e interfaces. Son usados para:

* Prevención de conflictos de nombres, por ejemplo, pueden haber clases con el mismo nombre siempre y cuando estén en diferentes paquetes.
* Permite tener un mayor orden de las clases de acuerdo a sus funcionalidades, por ejemplo, las clases del paquete javafx, son específicas para el desarrollo de aplicaciones de escritorio y las clases del paquete java Lang (paquete por defecto de Java) como la clase Math permiten realizar operaciones lógicas y aritméticas.

**Clase**: Modelo donde se redactan las características comunes de un grupo de objetos.

**Objeto:** Es un ejemplar o instancia de una clase, tiene propiedades(atributos) y comportamientos(métodos).

**Encapsulamiento:** Principio en el cual las propiedades y comportamientos de un objeto se mantienen o no al margen de otros objetos.

**Herencia:** Mecanismo de POO que permite la reutilización de código. Ejemplo: Considere que se desea crear un objeto Empleado, si antes se creo un objeto Persona, el objeto Empleado perfectamente puede heredar todos los atributos y comportamientos del objeto Persona ya que el objeto Empleado es una persona. A partir de ahí el nuevo objeto puede tener sus propios métodos y comportamientos como tener un salario, un ID, un jefe… cosas que un objeto Persona no tiene. Todas las clases heredan de la clase Object.

**Polimorfismo:** Principio de sustitución. Se puede utilizar un objeto de la subclase siempre que el programa espere un objeto de la superclase. Ejemplo si se desean conocer todas las características básicas de una persona (nombre, edad, país, fecha de nacimiento, altura, peso…) puedo llamar a dicho método Getter que lo realiza, sin embargo, en ese llamado perfectamente se puede indicar que en un lugar de dar las características básica de una persona, de las características de un empleado pues un empleado también es una persona.

**Enlazado dinamico:** Forma en la que Java distingue objetos que presentan o no herencia, relacionado con el polimorfismo.

**Clase Abstracta:** Clase que marca el diseno en la jerarquia de herencia. Normalmente es la clase mas generica de un programa con multiple herencia. Cuando se define un metodo abstracto (metodo que heredaran todas las subclases de esta superclase) la clase se convierte en una clase abstracta.

* Con que exista al menos un método abstracto, la clase que lo contiene debe declararse como una clase abstracta.
* Un método abstracto está obligado a definirse en cada subclase bajo la premisa de que todas las comparten, pero difieren en información. Ejemplo: imagine un método que devuelve la descripción general del objeto, un objeto persona, jefe y empleado deben heredar dicho método, sin embargo, la descripción general de cada uno de ellos varia ya que no son el mismo objeto, la información de dicho método varia.

**TEMA: UML y patrones de diseño**

**SDLS (Software Development Life Cycle):**

1. Recopilación de requerimiento
2. Análisis de requerimientos
3. Diseño de software
4. Implementación
5. Pruebas
6. Instalación y mantenimiento

Ingeniería de software: Código / Software de calidad (que funcione, sea mantenible, se ajuste al tiempo y presupuesto del proyecto, que sea reusable). La ingeniería de software esta titulada bajo SEI, ISO

Modelo de cascada – Waterfall: Crisis del software

Modelo en espriral: De este modelo nació Rational Unified Process que creo UML, otro es Agile (Screen, XP)

**Diseño Orientado a Objetos**

* Modelo para guiar al programador durante el proceso de diseño de software bajo el paradigma OO.
* Provee un conjunto de principios para calificar qué tan bien está el diseño de software.

**Primera Fase: Modelado Conceptual**

* Entender el dominio del problema y crear un modelo conceptual. Este modelo se puede hacer a criterio del diseñador
* Deber ser fácil de entender
* Storyboard
* ¿Cómo calza el software en el cuadro completo?

**Segunda Fase: Análisis**

* Construir historias de usuario / requerimientos
* User stories siguen un formato: “Yo como <rol> quiero ser capaz de <acción> con el fin de <objetivos>.”
* Se pueden crear prototipos para entender mejor el problema

**Tercera Fase: Arquitectura**

* Definir la estructura de solución para cumplir los requerimientos funcionales y de calidad
* Documentar las decisiones

**Cuarta Fase: Diseño detallado**

* Construir de forma iterativa un diseño detallado del sistema por construir
* Paso #1: Identifique las clases
  + Busque los sustantivos en las historias de usuario
  + Algunos sustantivos se convierten en clases. Otros se eliminan y otros se unen
  + Las clases deben tener una sola responsabilidad
* Paso #2: Identifique asociaciones
  + Identificar cómo interactúan las clases entre sí
  + Una asociación tiende a convertirse en un atributo
* Paso #3: Identifique atributos y métodos

**Patrones de diseño**

* Un patrón es una regularidad
* En el software hay regularidades o problemas recurrentes
* Reutilizar buenas soluciones es una buena práctica conocida por los ingenieros de software (basar nuevas soluciones en experiencias previas)

**Patrón de diseño:**

* Solución general para un problema común en un contexto dado
* Es una buena práctica
* Son conocidas por muchos profesionales a nivel mundial

**Hay 3 categorías de patrones de diseño**

* **Creacionales:**  Determinan cómo crear objetos
* **Estructurales:**  Cómo usar objetos entre sí mediante composición
* **De comportamiento:** Cómo comunicar objetos sin componerlos directamente

|  |  |  |
| --- | --- | --- |
| **Creacionales** | **Estructurales** | **De comportamiento** |
| Factory | Adapter | Intérprete |
| Builder | Bridge | Cadenas de responsabilidad |
| Singleton | Composite | Comando |
| Prototype | Decorator | Iterator |
|  | Facade | Memento |
|  | Proxy | Observer |
|  | Abstract Factory |  |

**Patrón Factory:** Crea una clase Factory que crea objetos concretos mediante un selector. El caller no conoce a las clases concretas.