UNViMe

Trabajo Práctico 0

Alumno: Tula E. Fernando

Carrera: Ing. de Sistemas de Información

Ejercicio 1

Considera un lenguaje que conozcas bien y analízalo en términos de los cuatro componentes de un paradigma mencionados por Kuhn.

Para resolver el ejercicio 1 analizaré el lenguaje Java en términos de los cuatro componentes de un paradigma:

1. Generalización simbólica: ¿Cuáles son las reglas escritas del lenguaje?

   En el caso de Java, las reglas escritas del programa son:

   - Un programa informático se crea en un proyecto.
   - Los proyectos contienen carpetas, entre las que se encuentran librerias y codigo, etc.
   - El programa se organiza en clases, las cuales son moldes o plantillas que al ser instanciadas en objetos , adquieren los atributos y métodos de la clase.
   - Los métodos son procedimientos o funciones que poseen los objetos instanciados.
   - Los atributos son los datos que posee un objeto. Los mismos pueden ser de tipo: privado, público, restringido y protegido.
   - Existen clases de tipos: hija, padre, hermana.
2. Creencias de los profesionales: ¿Qué características particulares del lenguaje se cree que sean "mejores" que en otros lenguajes?

   Este lenguaje es especial para aplicar el paradigma de programación orientada a objetos, de modo que a diferencia de lenguajes como C, que estan basados en programación estructurada, es mejor para encarar proyectos grandes, ya que con los pilares de la POO, que son 4, facilita mucho el trabajo, los cuales son:

   - Abstracción
   - Polimorfismo

- Encapsulamiento
- Herencia

3. Valores: ¿Qué pensamiento o estilo de programación consideraron mejor los creadores?

   En el caso de Java: Los creadores (James Gosling y el equipo de Sun Microsystems en los 90s) consideraban mejores ciertos valores:

   - Orientación a objetos como modelo mental principal: clases, objetos, herencia y polimorfismo como forma de organizar el software.
   - Portabilidad: que el mismo programa se pudiera ejecutar en cualquier máquina sin reescribirlo, gracias al lema "Write Once, Run Anywhere" y la Máquina Virtual de Java.
   - Simplicidad frente a C++: evitar complejidades como la aritmética de punteros, la gestión manual de memoria y herencia múltiple complicada.
   - Robustez y seguridad: que el lenguaje prevenga errores comunes (tipado fuerte, manejo obligatorio de excepciones, recolección de basura) y que los programas corran de forma más segura (muy pensado para aplicaciones distribuidas y la web).
   - Multithreading y redes: darle un lugar central al trabajo concurrente y la comunicación entre máquinas, algo que en los 90s estaba explotando.

4. Ejemplares: ¿Qué clase de problemas pueden resolverse más fácilmente en el lenguaje?

   Dado que java llegó con un nuevo paradigma de programación en los 90's, llegaba para solucionar las falencias de otros lenguajes, basados en paradigmas estructurados que presentan una serie de problemas. Por ello este lenguaje busco darles una solución con la implementación de un lenguaje modular, mantenible y con código legible, que permitiera el acoplamiento, escalable, con código reutilizable, con alta cohesión.

   Para ello busque ayude en la web para conocer con detalle cuando es más ventajoso el uso de este lenguaje:

   - Aplicaciones empresariales de gran escala: Sistemas de gestión, banca, logística, comercio electrónico. Java brilla aquí porque es robusto, orientado a objetos y tiene frameworks gigantes (Spring, Hibernate).
   - Aplicaciones multiplataforma: Gracias a la JVM, un mismo programa puede ejecutarse en Windows, Linux o Mac sin reescribir código.
   - Aplicaciones distribuidas y en red: Desde el principio Java se pensó para Internet: sockets, RMI (invocación remota de métodos), servidores web (Tomcat, JBoss).

- Aplicaciones con interfaces gráficas: Con bibliotecas como Swing o JavaFX, aunque hoy compite con otros lenguajes, Java fue muy usado para GUIs.
- Aplicaciones móviles (Android): Aunque Android usa una variante, gran parte de sus aplicaciones se escriben en Java.
- Problemas que requieren concurrencia (multithreading): Java incorporó desde sus primeras versiones un modelo robusto de hilos, facilitando la programación concurrente.

UNViMe

Ejercicio 2

Considera un lenguaje que conozcas bien y analízalo en términos de los ejes propuestos para la elección de un lenguaje de programación

1. ¿Tiene una sintaxis y una semántica bien definida? ¿Existe documentación oficial?

Java posee una sintaxis y una semántica bien definidas. Su sintaxis establece reglas estrictas sobre cómo escribir programas (declaración de clases, métodos, variables, estructuras de control), mientras que la semántica precisa el significado de esas construcciones en tiempo de compilación y ejecución. Además, el lenguaje cuenta con documentación oficial exhaustiva, principalmente en la Java Language Specification y en la Java API Documentation, mantenidas por Oracle y la comunidad de OpenJDK.

2. ¿Es posible comprobar el código producido en ese lenguaje?

Sí, en Java es posible comprobar el código. lo cual realiza en varias etapas:

- En primer lugar, el compilador verifica la sintaxis y el sistema de tipos antes de generar el bytecode.
- Posteriormente, la JVM realiza una verificación del bytecode para garantizar la seguridad y la corrección de las operaciones antes de ejecutar el programa.
- Además, existen herramientas y frameworks de prueba (como JUnit) que permiten comprobar de forma sistemática el comportamiento funcional del código.

3. ¿Es confiable?

Java es considerado confiable por varias razones:

- Tipado fuerte y verificación en compilación: evita operaciones incorrectas con los datos (no podés sumar un String con un int sin conversión explícita).
- Verificación de bytecode en la JVM: asegura que el código que se ejecuta no rompa las reglas del lenguaje.
- Manejo automático de memoria (Garbage Collector): reduce errores comunes como fugas de memoria o accesos a memoria liberada (problema típico en C o C++).
- Manejo de excepciones: obliga a tratar errores en tiempo de ejecución mediante try/catch, lo que mejora la robustez.
- Bibliotecas estándar confiables: gran parte de las funciones usadas (colecciones, IO, networking) ya están probadas y optimizadas.

- Independencia de plataforma: al correr sobre la JVM, el comportamiento es consistente sin importar el sistema operativo o hardware.

4. ¿Es ortogonal?

Java tiene un buen grado de ortogonalidad, aunque no es perfecto.

Puntos a favor:

- El sistema de tipos es consistente: los tipos primitivos y los objetos siguen reglas claras.
- Las estructuras de control (if, for, while, switch) funcionan de manera uniforme.
- Las clases, interfaces y herencia están bien definidas y se combinan de forma coherente.

Limitaciones:

- No todo es 100% ortogonal:
- Diferencia entre tipos primitivos (int, double) y objetos (no se tratan exactamente igual, aunque desde Java 5 existen los wrappers y el autoboxing que reducen esa brecha).
- No se permite herencia múltiple de clases, solo a través de interfaces:

  ¿Por qué Java no permite herencia múltiple de clases?

  Porque genera problemas de ambigüedad y complejidad.

  Ejemplo del famoso “problema del diamante”:

  - Clase A define un método m().
  - Clases B y C heredan de A.
  - Clase D hereda de B y C.

  Pregunta: ¿qué versión de m() debería usar D si B y C la redefinieron?

  Esto complica mucho la semántica y puede volver el código difícil de mantener. Java eligió herencia simple de clases + herencia múltiple de interfaces.

  - Una clase solo puede heredar de una superclase concreta (extends).

    - Pero puede implementar varias interfaces (implements).
    - Desde Java 8, las interfaces pueden tener métodos por defecto, lo que en la práctica da mucha flexibilidad sin los problemas del diamante.
- Algunos operadores no se pueden sobrecargar (a diferencia de C++).

5. ¿Cuáles son sus características de consistencia y uniformidad?

Java es un lenguaje uniforme ya que cuenta con los siguientes elementos:

- Reglas de sintaxis y semántica claras: todas las clases siguen el mismo esquema (declaración con class, métodos, atributos, modificadores).
- Orientación a objetos uniforme: todo en Java está basado en clases y objetos (salvo los tipos primitivos, que son la única excepción).
- Sistema de tipos coherente: tipado fuerte y verificaciones en compilación que funcionan de la misma manera en todo el lenguaje.
- Herencia simple + interfaces: decisión de diseño que mantiene consistencia, evitando problemas del diamante en herencia múltiple.
- Manejo de errores estandarizado: el sistema de excepciones usa siempre bloques try/catch/finally.
- Biblioteca estándar con criterios homogéneos: colecciones, clases de utilidades y librerías siguen convenciones similares (nombres, métodos, estructura).
- Portabilidad uniforme: gracias a la JVM, el mismo bytecode se ejecuta igual en distintas plataformas.

6. ¿Es extensible? ¿Hay subconjuntos de ese lenguaje?

En Java, sí:

- Bibliotecas y frameworks: podés ampliar la funcionalidad con librerías externas (Spring, Hibernate, JavaFX, etc.).
- Interfaces y clases abstractas: permiten que los programadores creen nuevas jerarquías y comportamientos sobre la base del lenguaje.
- Modularidad (Java 9+): con el Java Platform Module System se puede dividir el código en módulos reutilizables y escalables.
- Evolución del lenguaje: a lo largo de sus versiones, Java ha incorporado nuevas características (Generics en Java 5, lambdas en Java 8, records en Java 14, pattern matching en versiones más recientes).

7. El código producido, ¿es transportable?

En Java, la respuesta es sí, y de hecho es una de sus características más reconocidas:

- El código fuente en Java se compila a bytecode, un formato intermedio independiente de la máquina.
- Ese bytecode se ejecuta en la JVM (Java Virtual Machine), que actúa como capa de abstracción entre el programa y el sistema operativo/hardware.
- Gracias a esto, basta con tener una JVM adecuada instalada para que el mismo programa corra en Windows, Linux, Mac, o incluso dispositivos embebidos.
- Este principio se resume en el lema de Java: “Write Once, Run Anywhere”.

Ejemplo práctico: Si escribís un programa que imprime "Hola mundo" en Java y lo compilás, el .class generado funcionará igual tanto en una PC con Windows como en una con Linux, siempre que tengan JVM.