Jose Revete
19-10040

Examen 1

X=0, Y=4, Z=0

1. **Lenguaje escogido:** JAVA.
    a. **Diga qué tipo de alcances y asociaciones posee, argumentando las ventajas y desventajas de la decisión tomada por los diseñadores del lenguaje, en el contexto de sus usuarios objetivos.**

R: Alcance Estático y Asociación Superficial y Profunda (esto dependiendo del caso de uso). Recordando que los usuarios objetivos de JAVA es un rango bastante amplio dado que es un lenguaje de propósito general, podemos encontrar las siguientes:

Ventajas:

- Menor consumo de memoria, dado que solo ocupan un espacio en memoria compartido por todas las instancias.
- Mejor entendimiento visual en programas recursivos, porque se puede observar en el código de donde se toma el valor de, por ejemplo, una variable.
- Puede conducir a una ejecución más rápida, porque el alcance de las variables se puede determinar en tiempo de compilación, lo que elimina la necesidad de realizar búsquedas en tiempo de ejecución..

Desventajas:

- No es adecuado para implementar características dinámicas, dado que requieren un enfoque más flexible para el alcance variable, por ejemplo, el alcance dinámico o las clausuras.
- Puede dificultar la consecución de la encapsulación, un principio fundamental de POO, porque puede permitir un acceso no deseado a los datos desde cierto lugar del programa a otro, lo que desafía la protección y el control que busca la encapsulación.

    b. **Diga qué tipo de módulos ofrece (de tenerlos) y las diferentes formas de importar y exportar nombres.**

R:

- Módulos de Aplicación: Son aquellos que definen una aplicación Java modular. Se declaran en un archivo module-info.java y pueden exportar paquetes y requerir otros módulos.
- Módulos de Biblioteca: Proporcionan funcionalidades reutilizables que pueden ser utilizadas por otros módulos. También se declaran en un module-info.java.
- Módulos de Sistema: Son módulos que forman parte del JDK y son utilizados por la JVM. Incluyen módulos como java.base, java.logging, etc.
- Módulos de Terceros: Son módulos que provienen de bibliotecas externas o de terceros. Pueden ser utilizados en tu aplicación siempre que estén disponibles en el classpath.
- Módulos de Servicios: Permiten la implementación de un patrón de diseño de servicios, donde un módulo puede ofrecer servicios a otros módulos.
- Módulos de Prueba: Se utilizan para realizar pruebas unitarias y de integración de otros módulos. Generalmente, se organizan en un proyecto separado o en un submódulo.

Exportar:

1. Exportar un Paquete Completo: Esto permite que todos los tipos públicos dentro de paquete.a.exportar sean accesibles por otros módulos.

```
module nombre.modulo {
    exports paquete.a.exportar;
}
```

2. Exportar un Paquete Condicionalmente: Esto permite que el paquete sea accesible solo para el módulo específico otro.modulo.

```
module nombre.modulo {
    exports paquete.a.exportar to otro.modulo;
}
```

Importar:

1. Requerir un Módulo Completo: Esto indica que el módulo actual necesita el módulo otro.modulo para compilar y ejecutarse.

```
module nombre.modulo {
    requires otro.modulo;
}
```

2. Requerir un Módulo Condicionalmente: Esto significa que no solo el módulo actual requiere otro.modulo, sino que también cualquier módulo que requiera el módulo actual también tendrá acceso a otro.modulo.

```
module nombre.modulo {
    requires transitive otro.modulo;
}
```

3. Requerir un Módulo con Restricciones: Esto significa que otro.modulo es necesario solo en tiempo de compilación, no en tiempo de ejecución.

```
module nombre.modulo {
    requires static otro.modulo;
}
```

c. Diga si el lenguaje ofrece la posibilidad de crear aliases, sobrecarga y polimorfismo. En caso afirmativo, dé algunos ejemplos.

R:

- JAVA si tienes aliases, ejemplos:

  Ejemplo 1:

```
B[] b = new B[20];
A[] a = b;
a[0] =  new A();
B[0].fun1();
```

Ejemplo 2:

```
Rectangle b1 = new Rectangle (0, 0, 40, 120);
Rectangle b2 = b1;
```

- En JAVA no hay sobrecarga, solo hay una idea en Java similar a esta: la concatenación de cadenas con el operador más. Aparte de eso, Java no permite diseñar operadores propios.
- Si hay polimorfismo en JAVA, ejemplo:

```
class Bike{
  void run(){System.out.println("running");}
}
class Splendor extends Bike{
  void run(){System.out.println("running safely with 60km");}

  public static void main(String args[]){
    Bike b = new Splendor();//upcasting
    b.run();
  }
}
```

d. Diga qué herramientas ofrece a potenciales desarrolladores, como: compiladores, intérpretes, debuggers, profilers, frameworks, etc.

R:

- Compiladores:

  Javac: Compilador estándar de Java convierte el código fuente (.java) en bytecode (.class), que puede ejecutarse en cualquier máquina que tenga una Java Virtual Machine (JVM).
- Intérpretes:

  JVM (Java Virtual Machine): Actúa como intérprete del bytecode generado por el compilador. Es responsable de ejecutar las aplicaciones Java en diferentes plataformas, lo que permite la portabilidad ("write once, run anywhere").
- Depuradores (Debuggers):

  JDB (Java Debugger): Herramienta para depurar programas Java. Permite establecer puntos de interrupción, ver el estado de las variables, y seguir la ejecución paso a paso.

  Depuradores integrados en IDEs: IDEs como IntelliJ IDEA, Eclipse, y NetBeans ofrecen potentes herramientas gráficas de depuración que facilitan el seguimiento del flujo del programa y la identificación de errores.
- Profilers (Perfiles de rendimiento):

  Java VisualVM: Herramienta integrada en el JDK que permite analizar el rendimiento de aplicaciones Java, monitorear la memoria, el uso de CPU, identificar cuellos de botella y detectar fugas de memoria.

  JProfiler: Herramienta comercial con características avanzadas para análisis de rendimiento, memoria y depuración de hilos.

  YourKit Java Profiler: Profiler popular que ayuda a monitorear el rendimiento y el consumo de recursos de aplicaciones Java.
- Frameworks:

  Spring: Framework que ofrece infraestructura completa para el desarrollo de aplicaciones Java empresariales, microservicios, y aplicaciones web. Facilita la inyección de dependencias y el manejo de transacciones.

Hibernate: Framework de mapeo objeto-relacional (ORM) que facilita la interacción con bases de datos relacionales.

Apache Struts: Framework para el desarrollo de aplicaciones web basadas en MVC.

JavaServer Faces (JSF): Framework para la construcción de interfaces de usuario web en aplicaciones Java.

- Entornos de desarrollo integrados (IDEs):

  IntelliJ IDEA: IDE conocido por su inteligencia de código, facilidad de uso, y soporte para varios lenguajes de programación.

  Eclipse: IDE de código abierto que ofrece soporte para una amplia variedad de herramientas y lenguajes de programación.

  NetBeans: IDE de código abierto que es fácil de usar y soporta Java, además de otros lenguajes como PHP y C++.

- Herramientas de Build y Gestión de Dependencias:

  Maven: Herramienta de gestión de proyectos y dependencias. También automatiza el proceso de compilación, pruebas y generación de informes.

  Gradle: Herramienta de build moderna que combina lo mejor de Ant y Maven con un enfoque flexible y eficiente.

  Ant: Herramienta de automatización de compilaciones que permite definir procesos de construcción personalizados a través de archivos XML.

- Herramientas para Pruebas:

  JUnit: Framework estándar para pruebas unitarias en Java.

  TestNG: Framework de pruebas inspirado en JUnit, pero con características adicionales como soporte para pruebas paralelas y configuración más flexible.

  Mockito: Herramienta para crear mocks de objetos y realizar pruebas unitarias.

- Sistemas de Gestión de Versiones:

  Git: Sistema de control de versiones, ampliamente utilizado con herramientas como GitHub y GitLab para gestionar el código fuente de proyectos Java.

  Apache Subversion (SVN): Algunos proyectos Java más antiguos todavía utilizan SVN.

- Administración de Aplicaciones Java:

  JConsole: Herramienta que permite monitorear en tiempo real el rendimiento y comportamiento de aplicaciones que corren en la JVM, incluyendo el uso de memoria y hilos.

  JMX (Java Management Extensions): Proporciona una API estándar para la administración y monitoreo de aplicaciones empresariales.

- Documentación:

  Javadoc: Herramienta que permite generar documentación de API directamente desde el código fuente. Es estándar en la comunidad Java para generar documentación en formato HTML.

- Contenedores y Servidores de Aplicaciones:

  Apache Tomcat: Contenedor ligero para aplicaciones web basadas en servlets y JSP.

  JBoss/WildFly: Servidor de aplicaciones Java EE completo que soporta un conjunto completo de APIs empresariales de Java.

  Jetty: Servidor web y contenedor de servlets ligero, ideal para proyectos que requieren un servidor de aplicaciones sencillo y embebible.

- Monitoreo y Gestión de la JVM:

  Garbage Collection Logs: Registros detallados de las actividades del recolector de basura, útiles para ajustar el rendimiento.

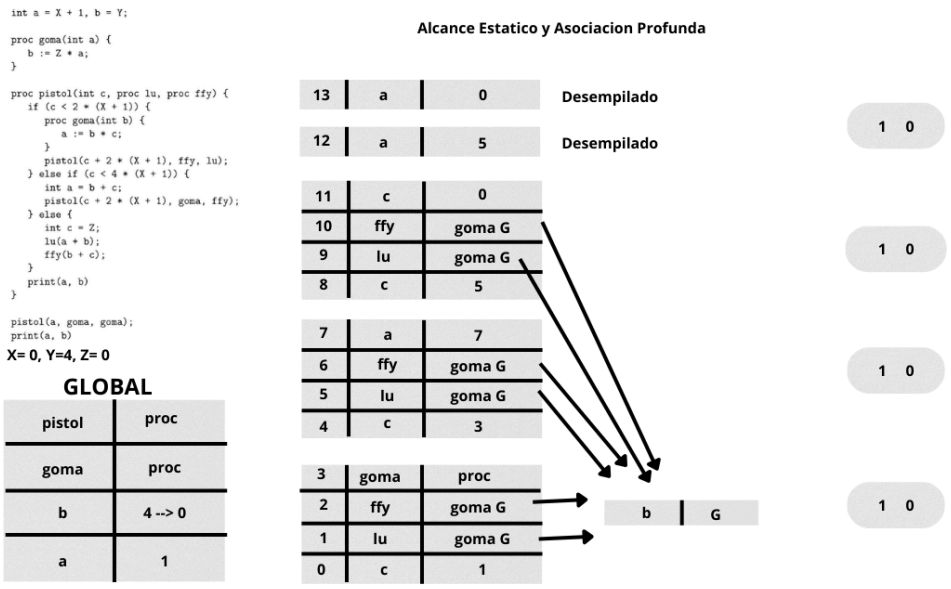
Jstat: Herramienta de monitoreo que permite observar el comportamiento de la memoria en tiempo real.

---

2. Considere el siguiente programa escrito en pseudo-código:

```
int a = X + 1, b = Y;

proc goma(int a) {
   b := Z * a;
}

proc pistol(int c, proc lu, proc ffy) {
   if (c < 2 * (X + 1)) {
      proc goma(int b) {
         a := b * c;
      }
      pistol(c + 2 * (X + 1), ffy, lu);
   } else if (c < 4 * (X + 1)) {
      int a = b + c;
      pistol(c + 2 * (X + 1), goma, ffy);
   } else {
      int c = Z;
      lu(a + b);
      ffy(b + c);
   }
   print(a, b)
}

pistol(a, goma, goma);
print(a, b)
```

a. <u>Alcance estático y asociación profunda:</u>



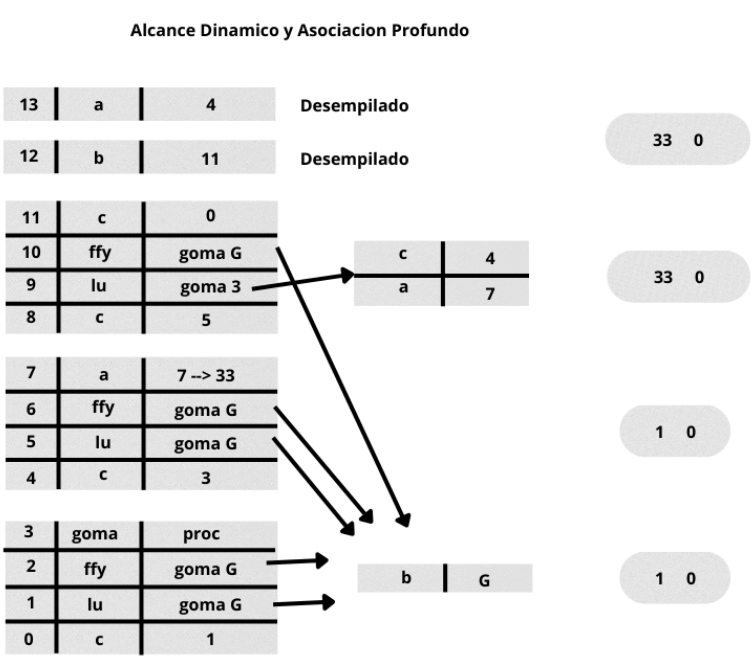
b. Alcance dinámico y asociación profunda:

```
int a = X + 1, b = Y;

proc goma(int a) {
   b := Z * a;
}

proc pistol(int c, proc lu, proc ffy) {
   if (c < 2 * (X + 1)) {
      proc goma(int b) {
         a := b * c;
      }
      pistol(c + 2 * (X + 1), ffy, lu);
   } else if (c < 4 * (X + 1)) {
      int a = b + c;
      pistol(c + 2 * (X + 1), goma, ffy);
   } else {
      int c = Z;
      lu(a + b);
      ffy(b + c);
   }
   print(a, b)
}

pistol(a, goma, goma);
print(a, b)
```

X= 0, Y=4, Z= 0

GLOBAL

pistol	proc
goma	proc
b	4 --> 0
a	1



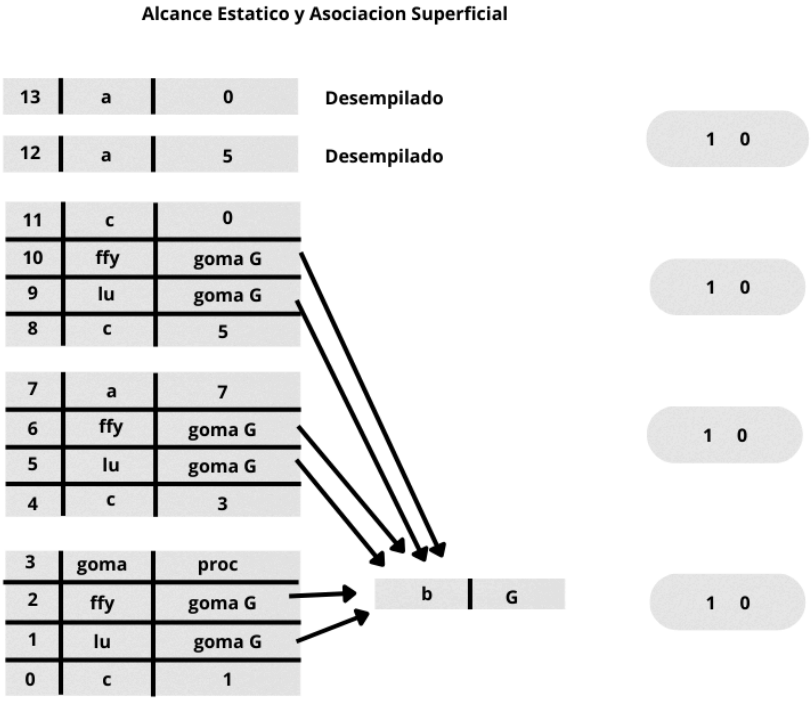
c. Alcance estático y asociación superficial:

```
int a = X + 1, b = Y;

proc goma(int a) {
   b := Z * a;
}

proc pistol(int c, proc lu, proc ffy) {
   if (c < 2 * (X + 1)) {
      proc goma(int b) {
         a := b * c;
      }
      pistol(c + 2 * (X + 1), ffy, lu);
   } else if (c < 4 * (X + 1)) {
      int a = b + c;
      pistol(c + 2 * (X + 1), goma, ffy);
   } else {
      int c = Z;
      lu(a + b);
      ffy(b + c);
   }
   print(a, b)
}

pistol(a, goma, goma);
print(a, b)
```

X= 0, Y=4, Z= 0

GLOBAL

pistol	proc
goma	proc
b	4 --> 0
a	1



d. Alcance dinámico y asociación superficial:

```
int a = X + 1, b = Y;

proc goma(int a) {
   b := Z * a;
}

proc pistol(int c, proc lu, proc ffy) {
   if (c < 2 * (X + 1)) {
      proc goma(int b) {
         a := b * c;
      }
      pistol(c + 2 * (X + 1), ffy, lu);
   } else if (c < 4 * (X + 1)) {
      int a = b + c;
      pistol(c + 2 * (X + 1), goma, ffy);
   } else {
      int c = Z;
      lu(a + b);
      ffy(b + c);
   }
   print(a, b)
}

pistol(a, goma, goma);
print(a, b)
```

X= 0, Y=4, Z= 0

GLOBAL

pistol	proc
goma	proc
b	4 --> 0
a	1