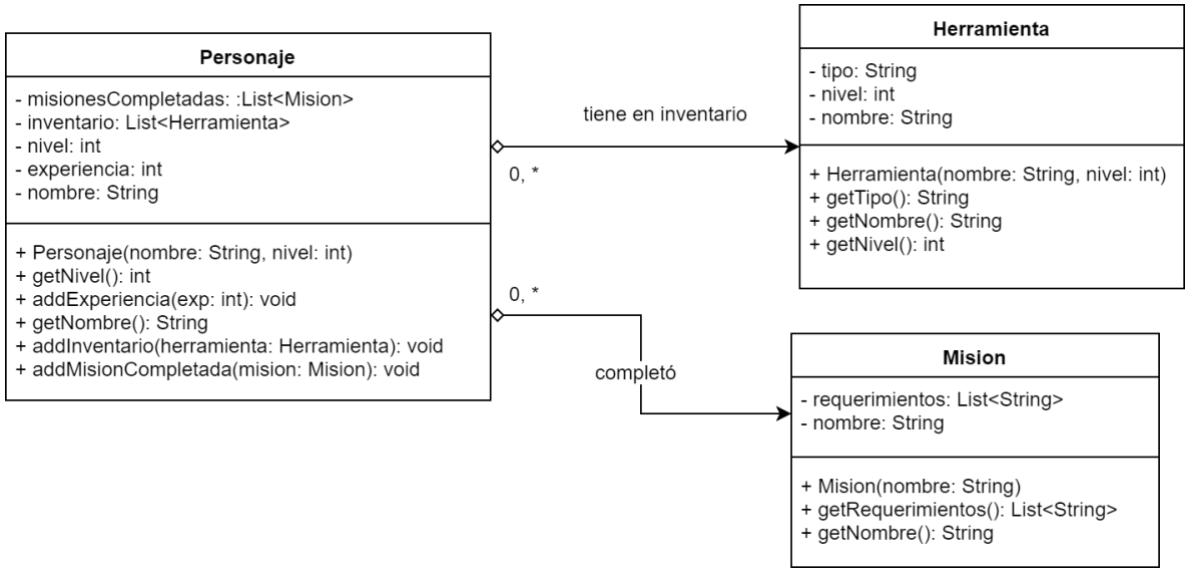
Examen 111Mil – Videojuego de Aventuras

Uno de los alumnos de 111Mil es un apasionado de los videojuegos. Con un grupo de compañeros decidieron poner en práctica lo aprendido durante el curso y diseñar e implementar su propio videojuego en lenguaje Java. El juego es muy simple y consta de un personaje que reúne herramientas en pos de cumplir una misión. A medida que los objetivos son completados, el personaje adquiere experiencia y sube de nivel, lo cual le permite utilizar mejores herramientas. Las misiones tienen un conjunto de restricciones que se condicen con el "tipo" de una herramienta. Por ejemplo, algunas misiones requieren herramientas de tipo "escalada", otras misiones requieren herramientas de "cocina" y otras de "apertura de candados o cerraduras". El personaje puede tener en su inventario herramientas que aún no puede utilizar. Esto sucede cuando el nivel de la herramienta es mayor al nivel actual del personaje.

Ejercicio 1. Implementar desde el diagrama de clases

El grupo de alumnos "Gamers 111Mil" realizó el diagrama UML de la aplicación e implementaron las clases *Herramienta* y *Misión*. Se nos pide que implementemos la clase *Personaje*. Para ello tenga en cuenta que:

a) La lista de misiones completadas y el inventario se deben inicializar vacías en el constructor. La experiencia se debe inicializar en 0.
b) El método addExperiencia solo suma experiencia, aún no se implementó el mecanismo para subir de nivel.



Ejercicio 2. Implementar un método a partir de un enunciado

Obtener una lista de herramientas del inventario dado un tipo de herramienta. Las herramientas retornadas deben tener un nivel menor o igual al del personaje (es decir, las debe poder utilizar). Recuerde indicar en qué clase se colocaría dicho método.

Ejercicio 3. Seguimiento de código

Considere el siguiente método de la clase Personaje:

```
      public int metodoSinNombre() {
  int res = 0;
  for (Iterator<Herramienta> it = inventario.iterator();
       it.hasNext();) {
    Herramienta h = it.next();
    if (h.getNivel() <= this.getNivel()) {
      res++;
    }
  }
  return res;
}
```

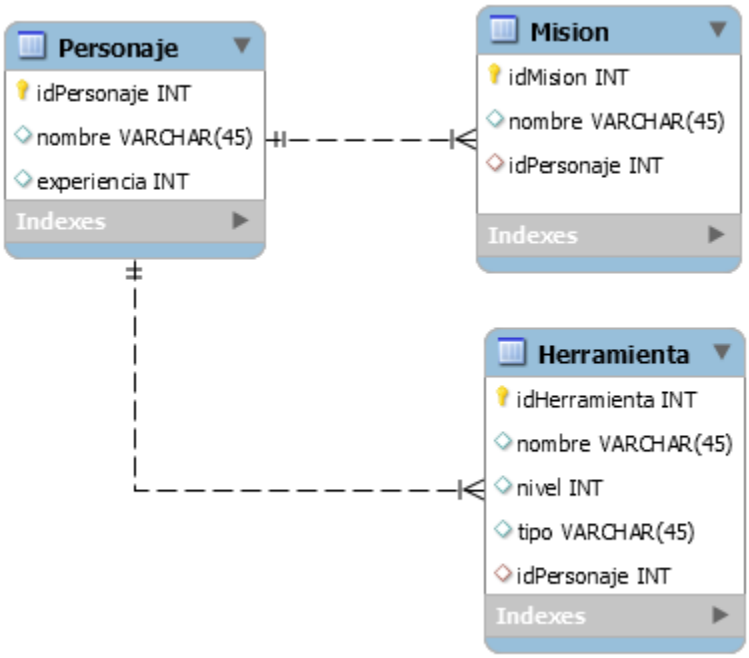
¿Qué imprimirá el programa al ejecutar el siguiente código?

```
public void main(String[] args) {
  Herramienta soga = new Herramienta("Soga", 4);
  Herramienta sarten = new Herramienta("Sartén", 1);
  Herramienta destornillador = new Herramienta("Destornillador", 10);

  Personaje p10 = new Personaje("111Mil", 10);
  p10.addInventario(destornillador);
  p10.addInventario(soga);
  System.out.println(p10.metodoSinNombre());

  Personaje p3 = new Personaje("111Mil", 3);
  p3.addInventario(soga);
  System.out.println(p3.metodoSinNombre());

  Personaje p0 = new Personaje("111Mil", 2);
  p0.addInventario(sarten);
  p0.addInventario(soga);
  p0.addInventario(destornillador);
  System.out.println(p0.metodoSinNombre());
}
```

Ejercicio 4. Javadoc

Elaborar la documentación técnica utilizando Javadoc del método *addMisionCompletada*.

Ejercicio 5. Consultas SQL



El grupo de desarrolladores desea armar una interfaz gráfica donde se muestre el inventario. En una de sus secciones quieren mostrar cuantas herramientas de cada tipo posee el Personaje con nombre “111Mil”. Hasta ahora, se cuenta con la siguiente consulta:

```
SELECT COUNT(*) FROM Herramienta;
```

Completar la consulta para obtener dicha información.

Ejercicio 6. Hibernate

Dado el diagrama de entidad-relación presentado en el ejercicio anterior y el diagrama UML presentado en el ejercicio 1, escribir el *mapping* de la clase *Personaje* a la tabla *Personaje* y mostrar cómo se declara el atributo “nombre”.

Soluciones

Ejercicio 1. Implementar desde el diagrama de clases

```
public class Personaje {
  private List<Mision> misionesCompletadas;
  private List<Herramienta> inventario;
  private int nivel;
  private int experiencia;
  private String nombre;

  public Personaje(String nombre, int nivel) {
    this.nombre = nombre;
    this.nivel = nivel;
    this.misionesCompletadas = new ArrayList<Mision>();
    this.inventario = new ArrayList<Herramienta>();
    this.experiencia = 0;
  }

  public int getNivel(){
    return this.nivel;
  }

  public addExperiencia(int exp){
    this.experiencia += exp;
  }

  public String getNombre(){
    return this.nombre;
  }

  public void addInventario(Herramienta herramienta) {
   this.inventario.add(herramienta);
  }

  public void addMisionCompletada(Mision mision) {
   this.misionesCompletadas.add(mision);
  }
}
```

Ejercicio 2. Implementar un método a partir de un enunciado

```
public List<Herramienta> obtenerHerramientas(String tipo){
  List<Herramienta> res = new ArrayList<Herramienta>();
  for (Iterator<Herramienta> it = this.inventario.iterator();
       it.hasNext(); ) {
    Herramienta h = it.next();
    if (h.getTipo().equals(tipo)
        && h.getNivel()<=this.getNivel()) {
      res.add(h);
    }
  }
  return res;
}
```

Ejercicio 3. Seguimiento de código

Imprime 2, 0 y 1.

Ejercicio 4. Javadoc

```
/**
* Agrega una misión completada a la lista de misiones completadas
* @param mision La misión completada
*/
public void addMisionCompletada(Mision mision) {
   this.misionesCompletadas.add(mision);
}
```

Ejercicio 5. Consultas SQL

```
SELECT h.tipo, COUNT(h.tipo) FROM Herramienta h
INNER JOIN Personaje p
ON (p.idPersonaje = h.idPersonaje)
WHERE p.nombre='111Mil'
GROUP BY h.tipo
```

Ejercicio 6. Hibernate

```
<class name = "Personaje" table = "Personaje">
  <property name = "nombre" column = "nombre" type = "string"/>
</class>
```