1. Explique las siguientes características de la programación orientada a objetos:

* Encapsulamiento
* Herencia
* Polimorfismo

1. Herencia

* Explique que tipo de herencia existe en Java y cual es la alternativa si deseamos agregar más funcionalidad compartida
* Cual es la diferencia principal entre clase abstracta y una interfaz en java
* En base al punto 2, menciona al menos dos diferencias clave en su implementación y uso

1. Defina los siguientes conceptos básicos de la POO:

* Clase
* Atributo
* Método

1. Excepciones en java:

* Explique que es una excepción en java
* Defina el uso del bloque try-cat ch-finally en un programa de java
* Cual es la diferencia entre las sentencias throw y throws en cuanto al manejo de excepciones

1. Interfaces graficas en Java

* Menciona las 3 librerias graficas de java y al menos una ventaja y una desventaja que posee cada una de ellas
* Explique el concepto de “escuchadores” al trabajar con interfaces graficas
* Mencione para que se usan las clases Stage y Scene en una aplicación de javaFx

1. Api de colecciones

* Explique que es un genérico en java y de un ejemplo de código de una línea mostrando su aplicación
* Explique la diferencia entre listas, conjuntos y mapas
* Mencione al menos un caso de uso optimo para los 3 tipos de colecciones mencionados en el punto anterior

1. Metodo construcor:

* Que es un constructor en java
* Puede una clase tener múltiples constructores
* Explique super()

1. Sobreescritura:

* Desarolle el concepto de sobreescritura
* Diferencia entre heredar y sobreescribir
* Porque no se recomienda que los objetos se comparen con == y cual es la alternativa sugerida

1. Relaciones entre clases:

* Explique la naturaleza de cada relación
* Q significa que tiene cardinalidad
* Un ejemplo sencillo del tipo de relación

**1. Características de la programación orientada a objetos**

* **Encapsulamiento:**  
  Es el principio que permite ocultar los detalles internos de una clase y exponer solo lo necesario mediante métodos públicos (getters/setters). Protege los datos internos de accesos indebidos y errores accidentales.
* **Herencia:**  
  Permite que una clase herede atributos y métodos de otra. Esto promueve la reutilización de código. En Java, se usa extends para heredar de una clase.
* **Polimorfismo:**  
  Permite que una misma operación se comporte de forma diferente en diferentes clases. Se logra mediante herencia y sobreescritura de métodos.

**2. Herencia**

* **Tipo de herencia en Java:**  
  Java permite **herencia simple**, es decir, una clase solo puede extender (heredar) de una única clase. No se permite herencia múltiple de clases.
* **Alternativa a la herencia múltiple:**  
  Se utilizan **interfaces** para implementar funcionalidades compartidas entre múltiples clases, permitiendo una forma de herencia múltiple de comportamiento.
* **Diferencia entre clase abstracta e interfaz:**
  + Una **clase abstracta** puede tener métodos implementados y atributos.
  + Una **interfaz** solo define métodos abstractos (aunque desde Java 8 puede tener métodos por defecto).
* **Dos diferencias clave:**
  + Una clase puede **implementar múltiples interfaces**, pero solo puede **extender una clase abstracta**.
  + Las clases abstractas pueden tener **constructores y atributos**, las interfaces no (aunque sí constantes public static final).

**3. Conceptos básicos de la POO**

* **Clase:**  
  Es un molde o plantilla que define las características (atributos) y comportamientos (métodos) de un objeto.
* **Atributo:**  
  Son las variables que definen el estado de un objeto.
* **Método:**  
  Son las funciones que definen el comportamiento de un objeto (lo que puede hacer).

**4. Excepciones en Java**

* **Qué es una excepción:**  
  Es un evento que interrumpe el flujo normal del programa cuando ocurre un error en tiempo de ejecución (como dividir por cero o acceder a un archivo inexistente).
* **Bloque try-catch-finally:**
  + try: contiene el código que puede lanzar una excepción.
  + catch: maneja la excepción lanzada.
  + finally: contiene código que siempre se ejecuta, haya o no excepción (como cerrar archivos).
* **Diferencia entre throw y throws:**
  + throw: se usa para lanzar una excepción.
  + throws: se declara en la firma del método para indicar que puede lanzar una o más excepciones.

**5. Interfaces gráficas en Java**

* **Tres librerías gráficas:**
  1. **AWT (Abstract Window Toolkit):**
     + Ventaja: parte del JDK original.
     + Desventaja: muy básica y dependiente del sistema operativo.
  2. **Swing:**
     + Ventaja: más flexible y completa que AWT.
     + Desventaja: interfaces pesadas y menos modernas.
  3. **JavaFX:**
     + Ventaja: moderna, permite usar CSS y FXML.
     + Desventaja: requiere instalación adicional en algunas versiones.
* **Escuchadores (Listeners):**  
  Son interfaces que se utilizan para manejar eventos (clics, teclas, movimientos del mouse). Detectan y responden a acciones del usuario.
* **Uso de Stage y Scene en JavaFX:**
  1. Stage: representa la ventana principal.
  2. Scene: contiene todos los elementos visuales que se muestran en el Stage.

**6. API de colecciones**

* **Qué es un genérico:**  
  Permite definir clases, interfaces y métodos con tipos de datos parametrizados.  
  Ejemplo:
* List<String> nombres = new ArrayList<>();
* **Diferencias entre Listas, Conjuntos y Mapas:**
  + **Lista (List):** mantiene el orden e incluye duplicados (ej: ArrayList).
  + **Conjunto (Set):** no permite duplicados (ej: HashSet).
  + **Mapa (Map):** almacena pares clave-valor (ej: HashMap).
* **Casos de uso óptimos:**
  + **List:** lista de productos en un carrito de compras.
  + **Set:** almacenar usuarios únicos registrados.
  + **Map:** buscar usuarios por su ID (clave-valor).

**7. Método constructor**

* **Qué es un constructor:**  
  Es un método especial que se ejecuta al crear un objeto, inicializando sus atributos.
* **¿Puede tener múltiples constructores?**  
  Sí, mediante **sobrecarga de constructores** (diferente cantidad o tipo de parámetros).
* **super():**  
  Se usa dentro de una subclase para llamar al constructor de la superclase.

**8. Sobreescritura**

* **Qué es la sobreescritura:**  
  Es redefinir un método de la superclase en la subclase para cambiar su comportamiento.
* **Diferencia entre heredar y sobreescribir:**
  + **Heredar:** usar directamente los métodos de la clase padre.
  + **Sobrescribir:** cambiar el comportamiento de un método heredado.
* **¿Por qué no usar == para comparar objetos?**  
  Porque == compara referencias en memoria, no contenido.  
  **Alternativa recomendada:** usar el método .equals().

**9. Relaciones entre clases**

* **Naturaleza de las relaciones:**
  + **Asociación:** una clase utiliza otra (ej: un Profesor da clases a un Alumno).
  + **Agregación:** relación débil de "todo-parte" (ej: una Escuela tiene Alumnos).
  + **Composición:** relación fuerte de "todo-parte", donde el todo controla la vida de las partes (ej: una Casa contiene Habitaciones que no existen fuera de ella).
  + **Herencia:** relación "es un", donde una clase deriva de otra.
* **¿Qué significa que tiene cardinalidad?:**  
  Indica la cantidad de instancias que puede tener una relación (ej: 1:1, 1:N, N:M).
* **Ejemplo sencillo:**
* class Persona {
* String nombre;
* }
* class Auto {
* Persona dueño; // Asociación: un Auto tiene un dueño
* }