1. **Asociación simple: La asociación simple es una relación entre dos clases en la que una clase utiliza o depende de la otra, sin una dependencia fuerte o permanente entre ellas. Una clase utiliza los métodos o atributos de la otra clase de manera ocasional.**

**Ejemplo de programa:**

**public class Estudiante {**

**private String nombre;**

**// Otros atributos y métodos**

**public void inscribirseEnCurso(Curso curso) {**

**// Lógica para inscribir al estudiante en un curso específico**

**}**

**}**

**public class Curso {**

**private String nombre;**

**// Otros atributos y métodos**

**}**

**En este ejemplo, la clase "Estudiante" tiene un método llamado "inscribirseEnCurso" que toma una instancia de la clase "Curso" como argumento. La clase "Estudiante" utiliza la clase "Curso" para realizar una acción específica, pero no hay una relación directa y exclusiva entre las instancias de ambas clases.**

1. **Asociación de composición: La asociación de composición es una relación en la que una clase contiene o "posee" a otra clase, y su existencia depende de la clase contenedora. La clase contenedora es responsable de crear y destruir las instancias de la clase contenida.**

**Ejemplo de programa:**

**public class Casa {**

**private Habitacion habitacionPrincipal;**

**// Otros atributos y métodos**

**public Casa() {**

**habitacionPrincipal = new Habitacion();**

**}**

**// Otros métodos**

**}**

**public class Habitacion {**

**// Atributos y métodos de la habitación**

**}**

**En este ejemplo, la clase "Casa" tiene un atributo llamado "habitacionPrincipal" que es una instancia de la clase "Habitación". La habitación principal solo existe mientras la casa exista, y se crea automáticamente cuando se crea una instancia de la clase "Casa".**

1. **Asociación de agregación: La asociación de agregación es similar a la asociación de composición, pero la clase contenida puede existir de forma independiente de la clase contenedora. La clase contenedora "agrega" instancias de la clase contenida sin ser responsable de su creación o destrucción.**

**Ejemplo de programa:**

**public class Universidad {**

**private List<Estudiante> listaEstudiantes;**

**// Otros atributos y métodos**

**public void agregarEstudiante(Estudiante estudiante) {**

**listaEstudiantes.add(estudiante);**

**}**

**// Otros métodos**

**}**

**public class Estudiante {**

**// Atributos y métodos del estudiante**

**}**

**En este ejemplo, la clase "Universidad" tiene una lista de estudiantes representada por el atributo "listaEstudiantes". La universidad puede agregar estudiantes a esta lista utilizando el método "agregarEstudiante". Los estudiantes pueden existir de forma independiente de la universidad y no se destruirán cuando se elimine la universidad.**

1. **Asociación inversa: La asociación inversa es una relación en la que una clase referencia a otra clase y viceversa. Ambas clases dependen entre sí.**

**Ejemplo de programa:**

**public class Padre {**

**private Hijo hijo;**

**// Otros atributos y métodos**

**}**

**public class Hijo {**

**private Padre padre;**

**// Otros atributos y métodos**

**}**

**En este ejemplo, la clase "Padre" tiene un atributo "hijo" que representa a su hijo, y la clase "Hijo" tiene un atributo "padre" que representa a su padre. Ambas clases se referencian mutuamente.**

1. **Asociación reflexiva: La asociación reflexiva es una relación en la que una clase está asociada consigo misma.**

**Ejemplo de programa:**

**public class Empleado {**

**private Empleado jefe;**

**// Otros atributos y métodos**

**public Empleado getJefe() {**

**return jefe;**

**}**

**public void setJefe(Empleado jefe) {**

**this.jefe = jefe;**

**}**

**// Otros métodos**

**}**

**En este ejemplo, la clase "Empleado" tiene un atributo "jefe" que representa al jefe de ese empleado. Un empleado puede tener otro empleado como jefe, lo que crea una relación reflexiva.**

1. **Enum: Un enum es una estructura de datos especial en Java que representa un conjunto de constantes. Los enums se utilizan para definir un conjunto finito de valores que una variable puede tomar.**

**Ejemplo de programa:**

**public enum DiasSemana {**

**LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO**

**}**

**public class EjemploEnum {**

**public static void main(String[] args) {**

**DiasSemana dia = DiasSemana.MARTES;**

**System.out.println("Hoy es " + dia);**

**}**

**}**

**En este ejemplo, se define un enum llamado "DiasSemana" que contiene los días de la semana como constantes. En el método "main", se declara una variable "dia" del tipo "DiasSemana" y se asigna el valor "MARTES". Luego, se imprime el valor de la variable, que muestra "Hoy es MARTES".**

1. **Genéricos: Los genéricos en Java permiten crear clases y métodos que pueden ser reutilizados con diferentes tipos de datos. Proporcionan una forma de parametrizar las clases y los métodos para que puedan funcionar con cualquier tipo de dato.**

**Ejemplo de programa:**

**public class Lista<T> {**

**private List<T> elementos;**

**public Lista() {**

**elementos = new ArrayList<>();**

**}**

**public void agregarElemento(T elemento) {**

**elementos.add(elemento);**

**}**

**// Otros métodos**

**}**

**public class EjemploGenericos {**

**public static void main(String[] args) {**

**Lista<String> listaString = new Lista<>();**

**listaString.agregarElemento("Hola");**

**listaString.agregarElemento("Mundo");**

**Lista<Integer> listaEnteros = new Lista<>();**

**listaEnteros.agregarElemento(10);**

**listaEnteros.agregarElemento(20);**

**}**

**}**

**En este ejemplo, se define una clase genérica llamada "Lista" que puede almacenar cualquier tipo de elemento. En el método "main", se crean instancias de "Lista" para cadenas de texto y enteros, y se agregan elementos a cada una de ellas.**

1. **UML: UML (Lenguaje de Modelado Unificado) es un lenguaje estándar utilizado para representar visualmente y documentar diferentes aspectos de un sistema de software. Se utiliza para modelar la estructura y el comportamiento de las clases y sus relaciones, así como otros elementos del sistema.**

**No es posible proporcionar un ejemplo de programa para UML, ya que UML es una notación gráfica utilizada para representar visualmente el diseño de software y no se traduce directamente en código. Se utilizan diagramas UML para representar clases, relaciones entre clases, métodos, atributos, etc., con símbolos y conectores específicos.**