**Temarios de Java.**

Collections.

- Beneficios de uso.

* Reduce el esfuerzo de programación.
* Aumenta la velocidad y calidad de los programas.
* Mejora la interoperabilidad entre APIs no relacionados.
* Reduce el esfuerzo de aprender y usar nuevas APIs.
* Fomenta la reutilización de software.

- Métodos de la interface Collection:

**add:** Para agregar un elemento a la colección.

**addAll:** agrega todos los elementos de una colección a otra.

**clear:** elimina todos los elementos de una colección.

**contains:** verifica si un elemento específico se encuentra en la colección.

**containsAll:** retorna true una colección contienetodos los elementos de una colección especifica.

**remove:** para eliminar un elemento de la colección.

**removeAll:** elimina de una colección todos los elementos de una colección específica.

**size:** permite saber cuántos elementos existen en la colección.

**toArray:** devuelve un array nuevo de longitud igual al número de elementos en la colección.

- Arquitectura de la interface Collection:

La arquitectura unificada para manipular las colecciones

contienen lo siguiente:

**Interfaces:** permite manipular colecciones independientemente del detalle de su representación.

**Implementaciones:** son implementaciones concretas de la

colecciones interfaces. Son estructuras de datos reutilizables.

**Algoritmos:** son métodos que realizan funciones útiles como búsquedas y ordenamientos.

- Descripción de las siguientes interfaces:

**Collection:** son objetos que agrupan múltiples elementos en una unidad.

**Set:** encargada del tratamiento de conjuntos en el api de Java.

**List:** encargada de agrupar una colección de elementos en forma de lista.

**Maps:** es una especie de diccionario donde a un objeto clave le asignamos un objeto valor.

**Queue:** es una estructura de datos que combina una cola y una pila ordinarias.

- Características ArrayList y LinkedList:

**ArrayList:**

* Muy rápida accediendo a elementos.
* Se adapta a un gran numero de escenarios.

**LinkedList:**

* Listas enlazadas.
* Gran eficiencia agregando y eliminando elementos.

- Características HashSet, LinkedHashSet y TreeSet:

**HashSet:**

* Rápida.
* No duplicado.
* No ordenamiento.
* No acceso aleatorio.

**LinkedHashSet:**

* Ordenación por entrada.
* Eficiente al acceder.
* No eficiente al agregar.

**TreeSet:**

* Es ordenado.
* Poco eficiente.

- Colas: Ventajas y desventajas:

**Ventajas:**

* Muy fácil de acceder al primer y ultimo elemento.
* Permite crear colas de elementos muy eficientes (LIFO/FIFO).

**Desventajas:**

* Acceso lento a los elementos intermedios.

- Colas de prioridad. Métodos. offer, poll, peek, clear, size:

**offer**: inserta un nuevo elemento en la cola si es posible.

**poll:** recupera y elimina un elemento del encabezado del. Devuelve nulo si la cola está vacía.

**peek:** recupera, pero no elimina, un elemento del principio de la cola. Devuelve nulo si la cola está vacía.

**clear:** Quita todos los objetos de la colección Queue.

**size:** Para conocer la cantidad de elementos.

- Métodos: get, put, remove, containsKey, containsValue, size, isEmpty.

**get:** Devuelve el valor para esa key.

**put:** Asocia el valor con la clave en el mapa.

**remove:** Elimina el elemento proporcionado por la clave.

**containsKey:** Nos devuelve valor false o true si existe la clave.

**containsValue:** Devuelve true si existe el valor.

**size:** Devuelve el número de elementos Clave/Valor del mapa.

**isEmpty:** Devuelve true si el mapa esta vacío.

- Maps: características:

**HashMap:**

* No ordenación.
* Eficiente.

**LinkedHashMap:**

* Ordenado por inserción.
* Permite ordenamiento por uso.
* Eficiente lectura.
* Poca eficiente escritura.

**TreeMap:**

* Ordenado por clave.
* Poco eficiente en todas sus operaciones.

**Hashtale:**

* Considerado obsoleto.
* Utilizado en operaciones de concurrencia.

Interfaz Stream, métodos:

From | stream()

Select | map()

Where | filter() (antes de un collecting)

order by | sorted()

distinct | distinct()

having | filter() (después de un collecting)

join | flatMap()

union | concat().distinct()

offset | skip()

limit | limit()

group by | collect(groupingBy())

count | count()

- Java Generics: Nombres de parámetros por convención.

* **E**: elemento de una colección
* **K**: clave
* **T**: tipo
* **V**: valor
* **S**, **U**, **V,** etc para otros tipos.

- Uso del Java Generics: Clases genéricas, Tipos genéricos, Métodos genéricos.

**Tipos Genéricos:**

* Se usan parámetros formales de tipo.
* Se ponen entre los símbolos < >
* Por convención se usa, una letra única, en mayúscula
* El alcance de los parámetros de tipo, es la clase entera.

**Métodos genéricos:**

Si las operaciones realizadas por varios métodos sobrecargados son idénticas para cada tipo de argumento, los métodos sobrecargados pueden codificarse en forma mas compacta y conveniente, mediante el uso de un método genérico.

**Clases Genéricas:**

Las clases genéricas proporcionan los medios para describir el concepto de cualquier clase en forma independiente de su tipo. Así, podemos crear instancias de objetos con tipos específicos de la clase genérica.

- Programación Concurrente: Proceso, Threads:

**Proceso:**

* Entorno de ejecución propio.
* Propio espacio de memoria, y recursos.

**Threads:** o procesos livianos.

* Existen dentro de un proceso.
* Cada proceso tiene al menos uno.
* Comparten los recursos del proceso (memoria, y archivos abiertos).

- Estados de un subproceso en programación concurrente: nuevo, ejecutable, en espera, en espera sincronizado, terminado, bloquead.

Nuevo

El programa inicia el subproceso

Ejecutable

Notify/NotifyAll

En espera (R-E)

Wait

Wait/Sleep

En espera sincronizado (R-E)

El intervalo expira

Notify/NotifyAll

Terminado (R)

Se completa la tarea

Adjuntar interrupción se completa E/S

Entra instrucción synchronized

Bloqueado (RRE)

Se completa petición de E/S

Adquiere bloqueo interrupt

- Pruebas unitarias: Pruebas Unitarias: pruebas unitarias, pruebas de integración, pruebas de rendimiento.

**Pruebas unitarias:** una prueba unitaria es un fragmento de código escrito por un desarrollador que ejecuta una funcionalidad específica en el código que se va a probar y afirma un determinado comportamiento o estado.

**Pruebas de integración:** una prueba de integración tiene como objetivo probar el comportamiento de un componente o la integración entre un conjunto de componentes.

**Pruebas de rendimiento:** las pruebas de rendimiento se utilizan para comparar componentes de software repetidamente. Su propósito es garantizar que el código bajo prueba se ejecute lo suficientemente rápido incluso si está bajo una carga alta.

- Webservices: diferencias entre Soap y Rest:

Ventajas REST

Pocas operaciones con muchos recursos

Se centra en la escalabilidad y rendimiento a gran escala para sistemas distribuidos hipermedia

HTTP GET, HTTP POST, HTTP PUT, HTTP DEL

XML auto descriptivo

Sincrono

HTTPS

Comunicación punto a punto y segura

Ventajas SOAP

Muchas operaciones con pocos

recursos

Se centra en el diseño de aplicaciones distribuidas

SMTP.HTTP POST, MQ

Tipado fuerte, XML Schema

Sincrono y Asincrono

WS SECURITY

Comunicación origen a destino

seguro

- Swing. Componentes Básicos.

JFrame: clase para generar ventanas sobre las cuales añadir distintos objetos con los que podrá interactuar el usuario

JPanel: Proporciona un área en la que pueden colocarse y organizarse los componentes. También puede utilizarse como un área de dibujo para gráficos.

JLabel: Muestra texto que no puede editarse, o iconos.

JTextField: Permite al usuario introducir datos mediante el teclado. También se puede utilizar para mostrar texto que puede o no editarse.

JComboBox: Proporciona una lista desplegable de elementos, a partir de los cuales el usuario puede realizar una selección, haciendo clic en un elemento o posiblemente escribiendo en el cuadro.

JButton: Activa un evento cuando se oprime mediante el ratón.

JCheckBox : Las casillas de verificación

JRadioButton : botones de opciones

JToggleButton : Llamado botón de activación, o switch

JTextArea : Es un área de texto de varias líneas

JPasswordField : Es un área de contraseña

JSpinner : Este componente es una caja para escribir números, donde con un par de botones que forman parte del mismo, incrementamos o decrementamos los valores contenidos.

JProgressBar : Es una barra de progreso

JList: Proporciona una lista de elementos a partir de los cuales el usuario puede realizar una selección, haciendo clic en cualquier elemento en la lista. Pueden seleccionarse varios elementos.

JScrollBar . Es una barra de desplazamiento

JSlider : Este componente llamado deslizador

- Modelo Relacional Vs Modelo orientado a objetos.

**El modelo relacional** es un modelo de datos basado en la lógica de predicado y en la teoría de conjuntos para la gestión de una base de datos.

**El modelo orientado a objetos** en una única entidad denominada objeto, se combinan las estructuras de datos con sus comportamientos. En este modelo se destacan conceptos básicos tales como objeto, clase y herencia.

Una de las diferencias se debe a que en los sistemas de bases de datos relacionales, los datos siempre se manejan en forma de tablas, formadas por un conjunto de filas o tuplas; mientras que en los entornos orientados a objetos los datos son manipulados como objetos, formados a su vez por objetos y tipos elementales.

- Que es ORM. Ventajas. Limitaciones. Asociaciones. Tipos de inicializaciones.

Es la técnica de programación que nos permite transformar los datos entre el sistema de tipos utilizados en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia. Lo que da como resultado una base de datos orientada a objetos virtual sobre la base de datos relacional.

Orm ventajas:

-Rapidez en el desarrollo. -Abstracción de la base de datos. -Reutilización.

-Seguridad. -Mantenimiento del código.

Orm limitaciones:

-Curva de aprendizaje lenta. -Menor rendimiento. -Sistemas complejos.

Orm Anotaciones:

@Entidad: Declara la clase como una entidad o una tabla.

@Tabla: Declara nombre de la tabla.

@Basic: Especifica no campos de restricción explícita.

@Embedded: Especifica las propiedades de la clase o de una entidad cuyo valor es una instancia de una clase se puede incrustar.

@Id: Especifica la propiedad, el uso de la identidad (la clave principal de una tabla de la clase.

@GeneratedValue: Especifica el modo en que la identidad se puede inicializar atributo como automática, manual o valor tomado de la tabla de secuencias.

@Transitorios: Especifica la propiedad que no es constante, es decir, el valor nunca se almacena en la base de datos.

@Columna: Especifica el atributo de columna para la propiedad persistence.

@SequenceGenerator Especifica el valor de la propiedad que se especifica en la anotación @GeneratedValue. Crea una secuencia.

@TableGenerator: Especifica el generador de valor para la propiedad

especificada en la anotación @GeneratedValue. Crea una tabla de generación de valor.

@AccessType: Este tipo de anotación se utiliza para establecer el tipo de acceso. Si establece el valor de @métodos Accesstype() y formattype() CAMPO), luego se produce acceso Campo sabio. Si establece el valor de @métodos Accesstype() y formattype() PROPIEDAD), a continuación, el acceso se produce bienes.

@JoinColumn: Especifica la entidad asociación o entidad colección. Esto se utiliza en muchos-a-uno y uno-a-muchas asociaciones.

@UniqueConstraint: Especifica los campos y las restricciones unique para la primaria o la secundaria.

@ColumnResult: Hace referencia al nombre de una columna de la consulta SQL que utiliza cláusula select.

@ManyToMany: Define una relación many-to-many entre el unir tablas

@Many ToOne: Define una relación de many-to-one entre el unir tablas

@OneToMany: Define una relación one-to-many entre los unir tablas

@OneToOne: Define una relación one-to-one entre los unir tablas

@NamedQueries: Especifica la lista de consultas con nombre.

@NamedQuery: Especifica una consulta con nombre estático.

Orm Asociaciones  
@ManyToOne Relation: Cada registro de una entidad hace referencia a muchas filas secundarias en otra entidad

@OneToMany Relación: Cada fila de una entidad hace referencia a muchos registros secundarios en otra entidad

@OneToOne Relation: Cada fila se refiere a una y solo una fila de otra entidad

Inicializadores:  
Lazy Fetch (lectura demorada): cuando existen relaciones entre distintas entidades, las que son dependientes de otras no sean inicializadas con sus valores almacenados en base de datos hasta que no sean explícitamente accedidas (leidas).

Lazy: @OneToMany y @ManyToMany. comportamiento por defecto

Eager Fetch: Es lo opuesto a Lazy, cargará por defecto TODAS las relaciones relacionadas con un objeto en particular.

- Herramientas que ayudan a evaluar el código fuente:

* SonarQube.
* Kritika.
* Klockwork.
* JArquitect.
* Crucible.