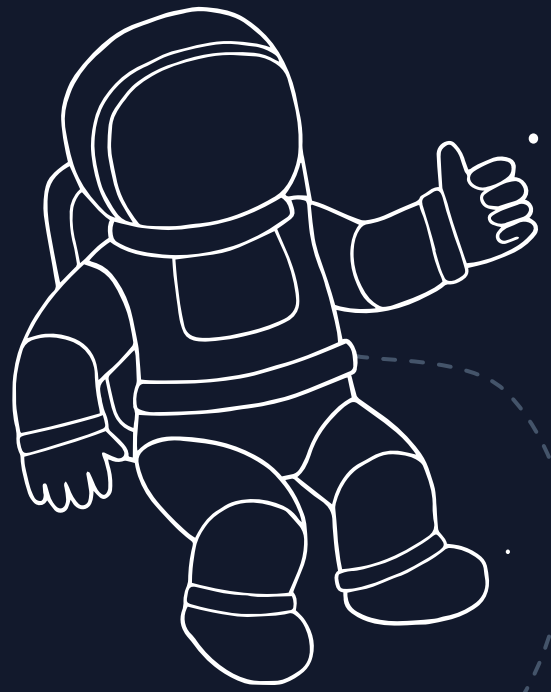
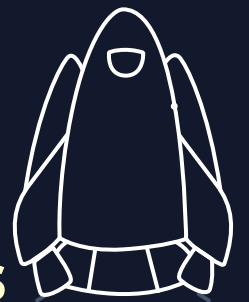


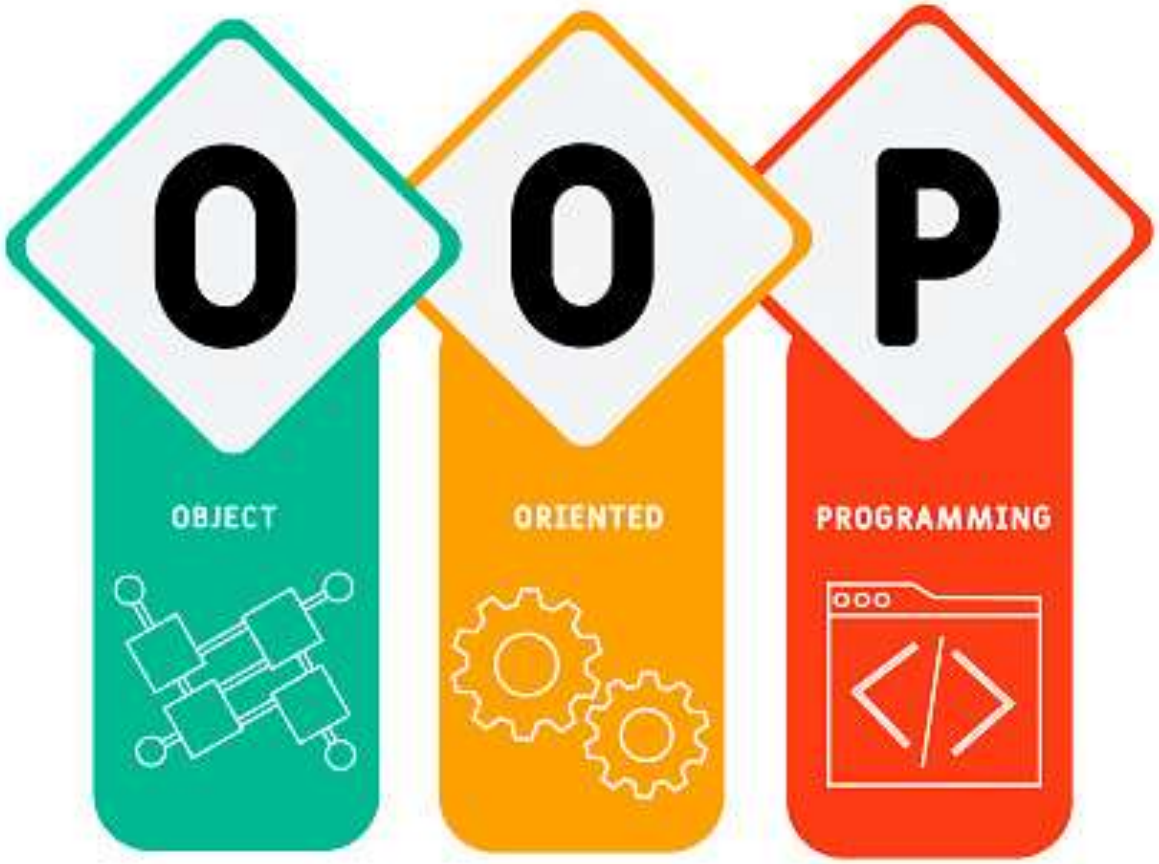
Programa académico CAMPUS

MODULO JAVA
Sesión 10
Conjuntos, mapas, enumeraciones y excepciones



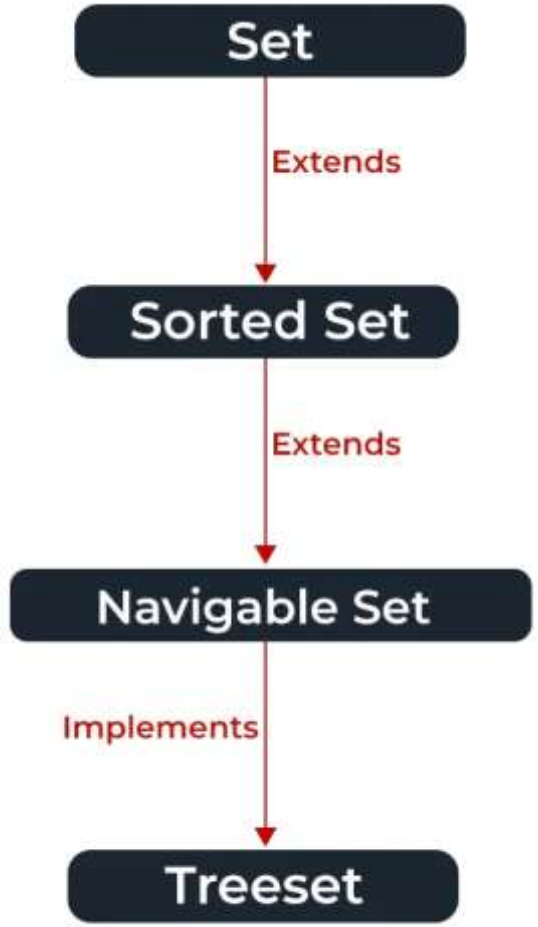
Trainer Carlos H. Rueda C.


Java™


O
O
P
OBJECT
ORIENTED
PROGRAMMING

Set - Conjuntos

- Es una interfaz de Java Collection Framework
- Extiende de la interfaz *Collection*
- Es una colección desordenada de objetos
- No se pueden almacenar valores desordenados
- Implementa el concepto matemático de conjunto
- Restringe la inserción de elementos duplicados
- la interfaz *NavigableSet* proporciona la implementación para navegar a través del conjunto
- La clase que implementa el conjunto navegable es *TreeSet*,

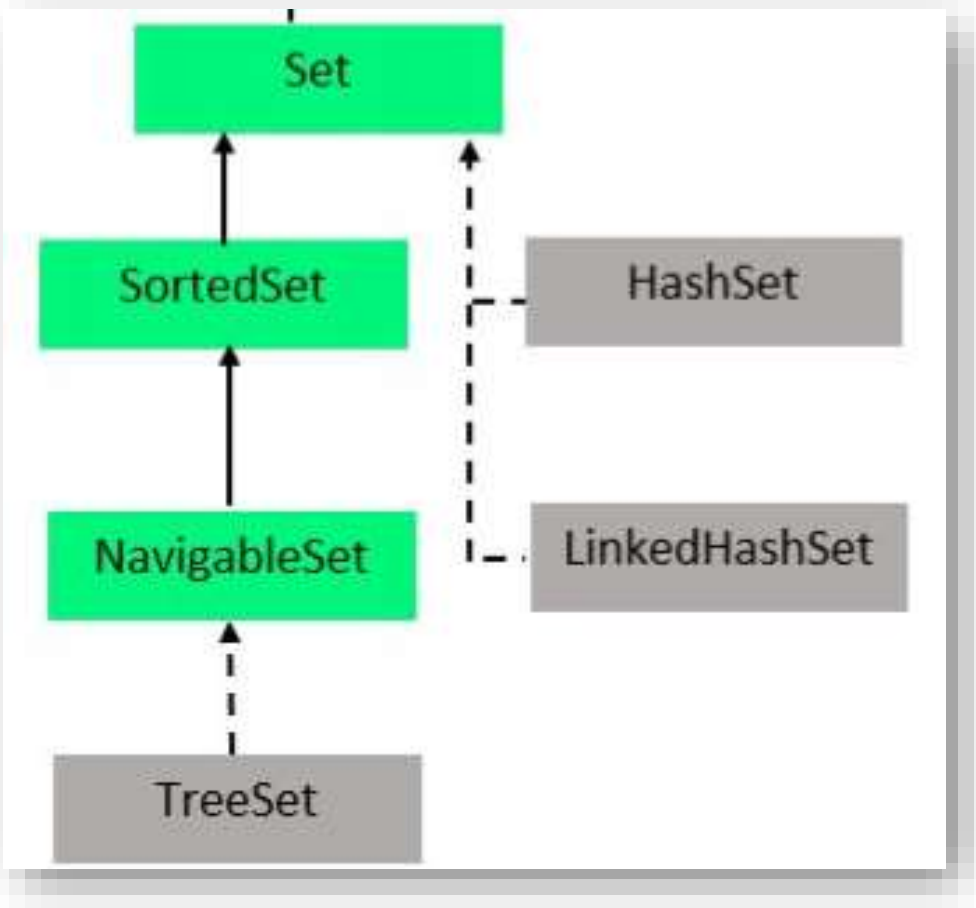


Set - Conjuntos

HashSet: Define el concepto de conjunto (grupo de elementos no repetidos) a través de una estructura Hash . Es el conjunto más habitual.

TreeSet: Define el concepto de conjunto a través de una estructura de Arbol . Este conjunto se utiliza en casos en los cuales necesitamos un **orden especifico de los elementos.**

LinkedHashSet: Define el concepto de conjunto añadiendo una lista doblemente enlazada en la ecuación que nos asegura que los elementos siempre se recorren de la misma forma.



Set – Declarar e Inicializar

Para declarar un conjunto en Java, se utiliza la siguiente sintaxis:

```
Set<TipoDeDato> nombreDeConjunto;
```

```
Set<Integer> nombreDeConjunto;
```

Después de declarar un conjunto, se necesita inicializarlo.

```
nombreDelArreglo = new TreeSet<Integer>();
```

Asignación de valores a un conjunto

```java
TreeSet<String> treeSet = new TreeSet<>();

// Agregar elementos al TreeSet
treeSet.add("Manzana");
treeSet.add("Banana");
treeSet.add("Naranja");
treeSet.add("Pera");

// Mostrar los elementos del TreeSet
System.out.println("Elementos en TreeSet: " + treeSet);
```

Recorrer elementos del conjunto

Con iteradores:

```
Iterator<Integer> iterator = treeSet.iterator();
while (iterator.hasNext()) {
  Integer elemento = iterator.next();
  System.out.println(elemento);
}
```

Con Estructura ForEach

```
for (Integer elemento : treeSet) {
  System.out.println(elemento);
}
```

Acceso a elementos del conjunto

Los conjuntos (Set) en Java no tienen un índice, ya que están diseñados para almacenar elementos únicos sin un orden específico. Sin embargo, puedes obtener un elemento en particular utilizando alguna de las siguientes estrategias

Convertir a una lista (más directo):

```
Set<String> set = new HashSet<>();
set.add("Elemento1");
set.add("Elemento2");
set.add("Elemento3");

List<String> lista = new ArrayList<>(set);
String elementoDeseado = lista.get(1); // Por ejemplo, el segundo elemento
System.out.println("Elemento obtenido: " + elementoDeseado);
```

Recorrer elementos del conjunto

Recorrer hasta encontrar el elemento deseado:

```
for (String elemento : set) {
    if (elemento.equals("Elemento2")) { // Cambiar según tu criterio
        System.out.println("Elemento encontrado: " + elemento);
        break;
    }
```

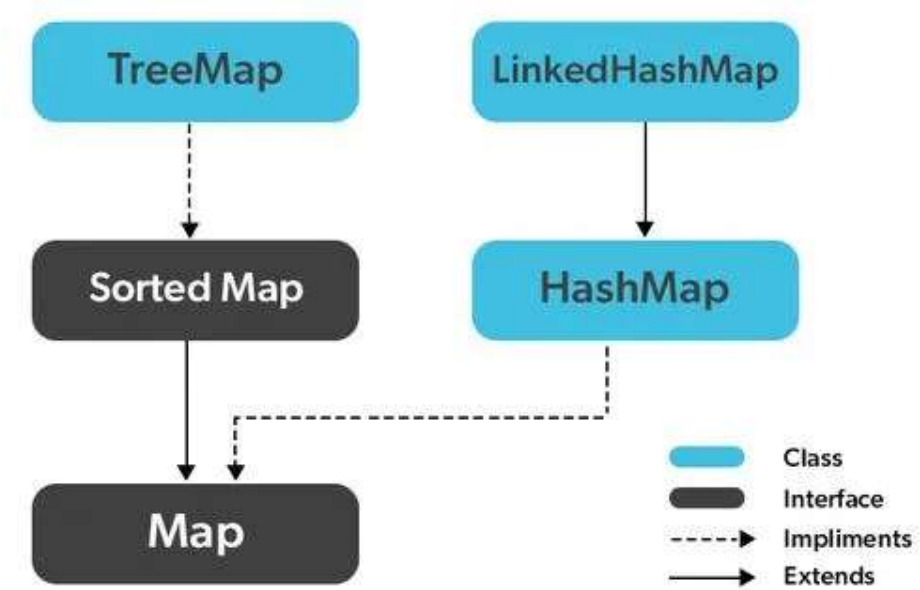
Método toString()

El método `toString()` convierte un conjunto en una representación de cadena legible. Este método es útil cuando se requiere imprimir o mostrar el contenido de un conjunto.

```
String resultado = treeSet.toString();
```

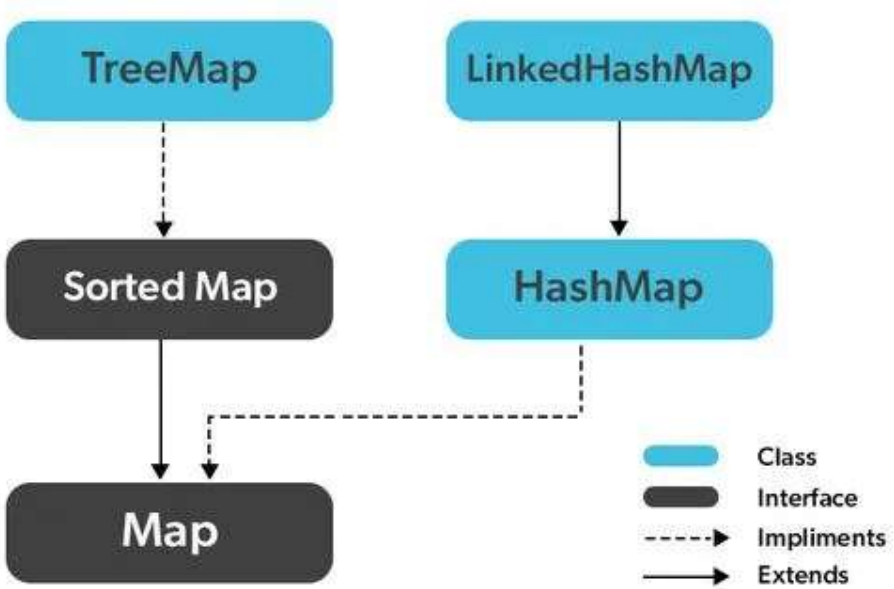
MAP

- Representa una asociación entre una clave y un valor
- La interfaz *Map* en Java no es un subtipo de la interfaz *Collection*, por lo que tiene un comportamiento un poco diferente de los demás tipos de colecciones.
- Un mapa contiene claves únicas.
- Los mapas son ideales para asociaciones de clave-valor, como diccionarios.



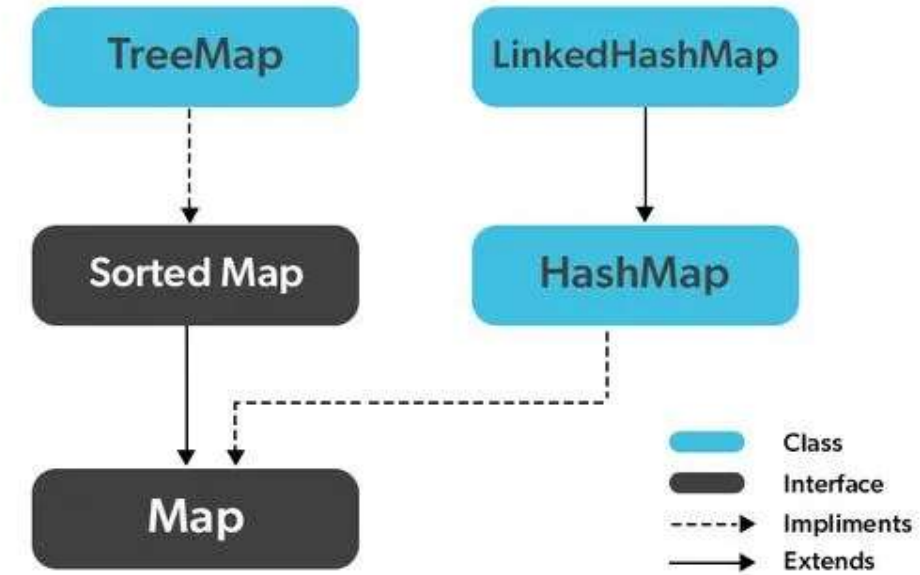
MAP

- **HashMap:** No garantiza ningún orden, ni de inserción ni natural.
- Generalmente, es la implementación más utilizada debido a su ***velocidad*** en casos donde ***el orden no importa***.



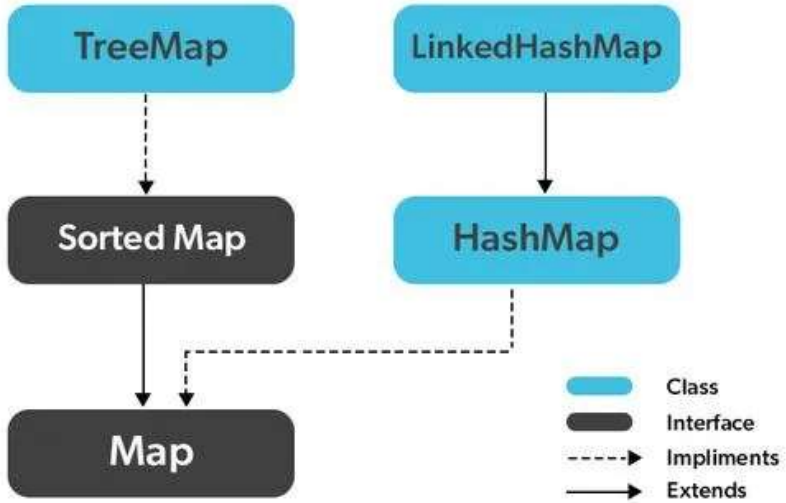
MAP

- **LinkedHashMap:** Mantiene el orden de inserción o el orden de acceso (si está configurado para ello).
- Cuando necesitas ***el orden de inserción o de acceso***, como en cachés o estructuras donde el orden importa.



MAP

- **TreeMap:** Ordena las claves en su orden natural o mediante un comparador proporcionado.
- Cuando necesitas ***claves ordenadas*** para iterar en un rango o para búsquedas específicas basadas en el orden.
- Proporciona una ***colección ordenada*** de pares clave-valor, donde las claves se ordenan ***según su orden natural*** o un ***Comparador*** personalizado *pasado al constructor*.



Declaración e inicialización de Mapas

```
nombreDeMapa = new HashMap<tipoDeDatoLlave, tipoDeDatoValor>();
```

```
Map<Integer, String> nombreDeMapa = new HashMap<Integer, String>();
```

Asignación de valores a un mapa y obtener valores a partir de claves

```java
Map<String, Integer> hashMap = new HashMap<>();

// Asignar valores al mapa usando el método put()
hashMap.put("Juan", 25);
hashMap.put("María", 30);
hashMap.put("Luis", 28);
hashMap.put("Ana", 35);

// Acceder y mostrar los valores del HashMap
System.out.println("Edad de Juan: " + hashMap.get("Juan"));
System.out.println("Edad de María: " + hashMap.get("María"));
System.out.println("Edad de Luis: " + hashMap.get("Luis"));
System.out.println("Edad de Ana: " + hashMap.get("Ana"));
```

Acceso a elementos de un mapa recorriéndolo

```
Iterator<Map.Entry<String, Integer>> iterator = hashMap.entrySet().iterator();
while (iterator.hasNext()) {
  Map.Entry<String, Integer> entry = iterator.next();
  String clave = entry.getKey();
  Integer valor = entry.getValue();
  System.out.println("Nombre: " + clave + ", Edad: " + valor);
}
```

Acceso a elementos de un mapa recorriéndolo

```
for (Map.Entry<String, Integer> entry : hashMap.entrySet()) {
  String clave = entry.getKey();
  Integer valor = entry.getValue();
  System.out.println("Nombre: " + clave + ", Edad: " + valor);
}
```

Tamaño y toString() de un Mapa

```
int tamaño = hashMap.size();
```

```
String representacionCadena = hashMap.toString();
```

Enumeraciones

Una enumeración, también conocido como `Enum`, es una clase especial que restringe la creación de objetos a los valores específicamente definidos en su implementación. La principal diferencia con una clase normal es que si un enumerado tiene un constructor, este debe ser privado para evitar la creación de nuevos objetos fuera de la enumeración.

```
public enum Pais {
    ESPANA, FRANCIA,ITALIA,ALEMANIA, REINO_UNIDO;
}
```

En este ejemplo se define una enumeración llamada `Pais` que puede tomar los valores descritos. Entonces por ejemplo si se requiere asignar el valor de Francia a una variable se hace de la siguiente forma:

```
Pais francia = Pais.FRANCIA;
```

Enumeraciones
Métodos Heredados de Enum

```
public enum Pais {
    ESPANA, FRANCIA,ITALIA,ALEMANIA, REINO_UNIDO;
}
```

```
francia.name();     // Devuelve un String con el nombre de la constante (FRANCIA)
francia.toString();     // Devuelve un String con el nombre de la constante (FRANCIA)
francia.ordinal();     // Devuelve un entero con la posición del enum según está declarad
francia.values();     // Devuelve un array que contiene todos los valores del enum
```

Enumeraciones

Restringe la creación de objetos a los definidos en su propia clase (por eso su *constructor es privado*, como se muestra en el siguiente fragmento de código). Sin embargo, estos objetos *pueden tener atributos* como cualquier otra clase.

```
public enum Pais {
    ESPANA("España", "Madrid"),
    FRANCIA("Francia", "París"),
    ITALIA("Italia", "Roma"),
    ALEMANIA("Alemania", "Berlín"),
    REINO_UNIDO("Reino Unido", "Londres");

    // Variables de instancia para nombre y capital del país
    private String nombre;
    private String capital;

    // Constructor privado para inicializar las variables de instancia
    private Pais(String nombre, String capital) {
        this.nombre = nombre;
        this.capital = capital;
    }

    // Método para obtener el nombre del país
    public String getNombre() {
        return nombre;
    }

    // Método para obtener la capital del país
    public String getCapital() {
        return capital;
    }
}
```

Enumeraciones

```java
public class MiPrimerProyecto {
    public static void main(String[] args) {
        // Acceder a las constantes de la enumeración
        Pais miPais = Pais.ESPANA;

        // Obtener información del país
        String nombrePais = miPais.getNombre();
        String capitalPais = miPais.getCapital();

        // Imprimir información del país
        System.out.println("País: " + nombrePais);
        System.out.println("Capital: " + capitalPais);
    }
}
```

Enumeraciones

Las enumeraciones (`enum`) en Java son útiles cuando necesitas definir un conjunto fijo y bien conocido de constantes relacionadas, con comportamientos o propiedades asociados. Son especialmente útiles en casos donde:

1. **Valores constantes bien definidos:** Un grupo limitado de opciones posibles, como días de la semana, meses, estados de un proceso, etc.
2. **Asociación de comportamiento:** Permitir que cada constante tenga métodos o propiedades específicas.
3. **Seguridad de tipo:** Garantizar que solo los valores válidos puedan ser utilizados.
4. **Legibilidad y mantenimiento:** Reemplazar cadenas o números mágicos por identificadores claros y significativos.
5. **Extensibilidad:** Incluir lógica específica para cada constante.

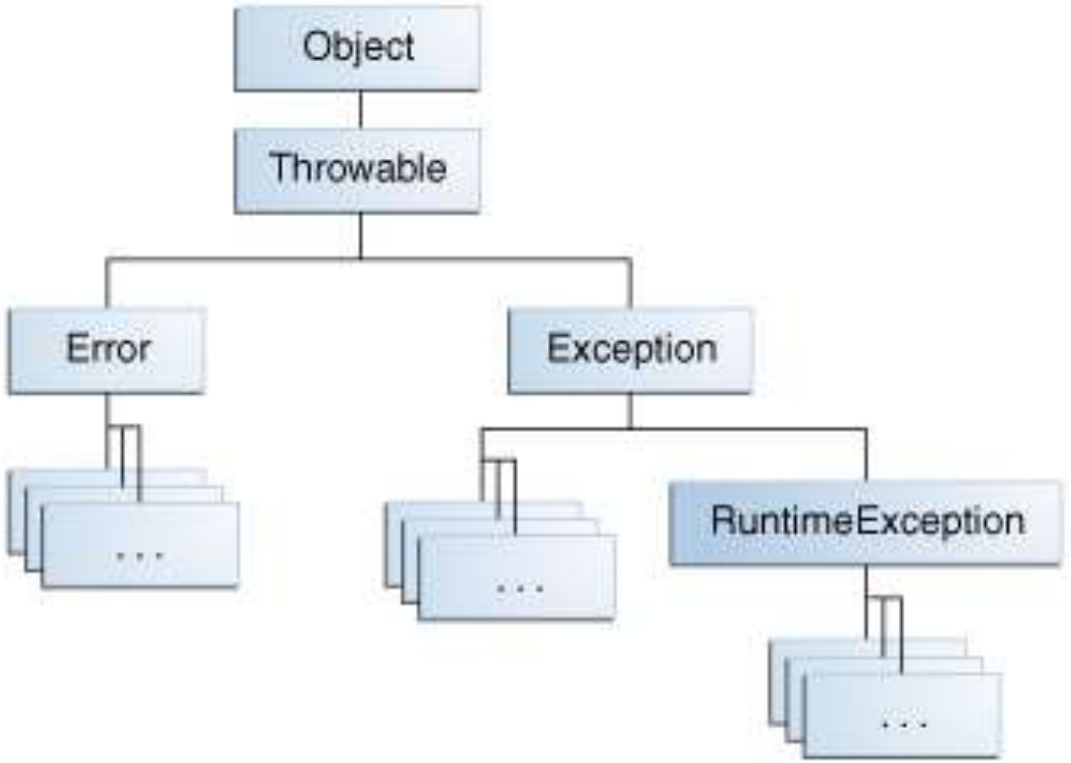
Excepciones

Las excepciones en Java son mecanismos cruciales para manejar errores y problemas durante la ejecución del programa. Se utiliza la estructura `try-catch` para capturar y gestionar estas excepciones en tiempo de ejecución. Dentro del bloque `try`, se coloca el código propenso a errores, que podría generar una excepción. Si se produce un error, la excepción se lanza y se puede capturar utilizando bloques `catch` específicos para tipos de excepciones particulares.

Exception

Excepciones

Además, existe un bloque opcional llamado "`finally`" que se ejecutará sin importar si se produce una excepción o no. También es importante señalar que en Java, las excepciones son objetos que derivan de la clase `Exception`, que a su vez es una subclase de la clase `Throwable`. Cuando un método debe lanzar una excepción, ya sea por nuestra elección o por requisitos del propio método, se puede usar la palabra clave "`throws`" seguida del nombre de la excepción que el método o función podría lanzar. Esto permite una gestión eficaz y segura de los problemas que puedan surgir durante la ejecución del programa.



Excepciones

```
try {
  //Sentencias de código que podrían generar un error
}
// En caso de que surja algún error
catch (Exception ex) {
  //Sentencias de código a ejecutar si se produce la excepción
}
finally {
  // Tanto si se produce la excepción, como si no se produce, se ejecutará el bloque
}
```

```java
double primerNumero, segundoNumero, resultado;
Scanner lectura=new Scanner(System.in);

try {
  System.out.println("Introduce el primer número");
  primerNumero=lectura.nextDouble();

  System.out.println("Introduce el segundo número");
  segundoNumero=lectura.nextDouble();

  resultado=primerNumero/segundoNumero;

  System.out.println("El resultado es " + resultado);
}
catch (ArithmeticException e){
  System.out.println("No se puede dividir entre cero");
}
catch (Exception ex){
  System.out.println("Se ha producido un error");
}
finally {
  lectura.close();
}
```

Bloques `Catch` :

- Si ocurre una excepción de tipo `ArithmeticException` (que se produce cuando se intenta dividir entre cero), el programa imprime "No se puede dividir entre cero".
- Si ocurre cualquier otra excepción (capturada como `Exception`), el programa imprime "Se ha producido un error".

```java
double primerNumero, segundoNumero, resultado;
Scanner lectura=new Scanner(System.in);

try {
  System.out.println("Introduce el primer número");
  primerNumero=lectura.nextDouble();

  System.out.println("Introduce el segundo número");
  segundoNumero=lectura.nextDouble();

  resultado=primerNumero/segundoNumero;

  System.out.println("El resultado es " + resultado);
}
catch (ArithmeticException e){
  System.out.println("No se puede dividir entre cero");
}
catch (Exception ex){
  System.out.println("Se ha producido un error");
}
finally {
  lectura.close();
}
```

Bloque `Finally` :

- El bloque `finally` se ejecuta siempre, sin importar si ocurre una excepción o no. En este caso, se cierra el objeto `Scanner` para liberar los recursos.

Excepciones más frecuentes en Java

`ArithmeticException`	Excepción de una operación aritmética (Ej. división por cero)
`ArrayStoreException`	Excepción en una operación en el uso de un array
`NegativeArraySizeException`	Excepción en una operación en el uso de un array
`NullPointerException`	Excepción producida por una referencia a un objeto nulo
`EOFException`	Excepción producida por alcanzar el final de un fichero
`IOException`	Excepción en un proceso de entrada y salida
`FileNotFoundException`	Excepción por no encontrar un fichero
`NumberFormatException`	Excepción en la conversión de una cadena de caracteres a un valor numérico

Como crear excepciones propias en Java

Para crear excepciones propias en Java basta con:

- Crear una clase que herede de la clase `Exception`.
- Crear métodos constructores para crear la excepción cuando se necesite
- Sobre escribir el método `getMessage` de la clase `Exception` para devolver el error.

```
//.......................
try {
    //.......................
    //.Lanza la Excepción.....
    //.......................
} catch (Exception exception) {
    //.......................
    //.Atrapa la Excepción.....
    //.......................
} finally {
    //.......................
    //.Liberación
```

Como crear excepciones propias en Java

```java
public class ExcepcionPersonalizada extends Exception{
    private String mensajeDeError;

    // Creación de un constructor sin parametros con asignación de valor por defecto a mensajeDeError
    public ExcepcionPersonalizada() {
        this.mensajeDeError="Error 1";
    }

    // Creación de un constructor con parametro para mensaje de error personalizado
    public ExcepcionPersonalizada(String ErrorPersonalizado) {
        this.mensajeDeError=ErrorPersonalizado;
    }

    // SE sobre escribe el metodo getMessage de la clase Exception
    @Override
    public String getMessage() {
        return mensajeDeError;
    }
}
```

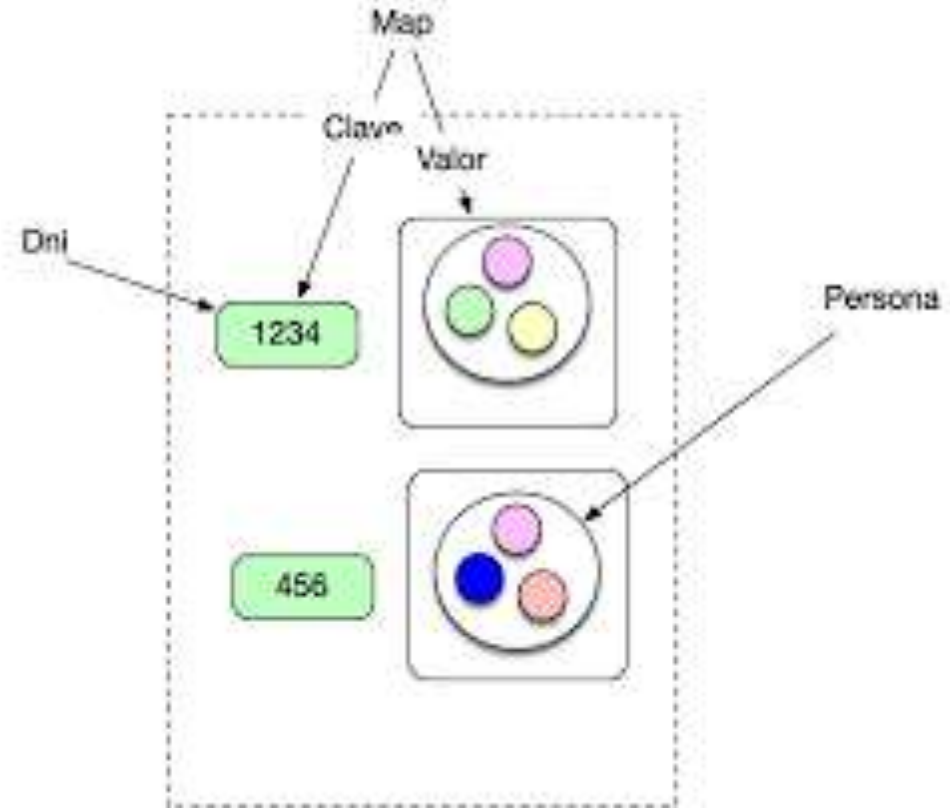
Como crear excepciones propias en Java

```
public static void main(String[] args) {
    Scanner lectura = new Scanner(System.in);
    String num1;
    try {
        System.out.println("Introduce el número");
        num1 = lectura.nextLine();
        if (num1.isEmpty()) {
            throw new ExcepcionPersonalizada();
        }
    }
    catch (ExcepcionPersonalizada e) {
        System.out.println(e.getMessage());
    } finally {
        lectura.close();
    }

}
```

Ejercicios

- Consulte los métodos más usados en las clases de conjunto y mapas
- Construya aplicaciones donde ejemplifique el uso de estos métodos



Ejercicios

Dada la siguiente información sobre las calificaciones de estudiantes de una institución educativa:

- Código
- Nombre
- Nota 1 (Peso de 30%) – *nota: [0, 10]*
- Nota 2 (Peso de 30%) – *nota: [0, 10]*
- Nota 3 (Peso de 40%) – *nota: [0, 10]*

El proceso se termina cuando el usuario indique que no quiere continuar (S/N)

Se pide calcular:

- ✓ Mostrar en orden la terna de los mejores estudiantes con sus nombres y nota final
- ✓ Mostrar cómo está el estudiante en relación al grupo por medio de un gráfico de barras.
- ✓ El rango de la cantidad de asteriscos es [0, 20]

Nota:

- ➢ Use diccionarios para gestionar la clase Estudiante
- ➢ Ejemplo del diagrama de barras:
- ➢ Maneje las excepciones que puedan ocurrir

```
Estudiante  Barras
-------------------------
1001          **
1002          ********
1003          **********
1004          *******
```

Ejercicios

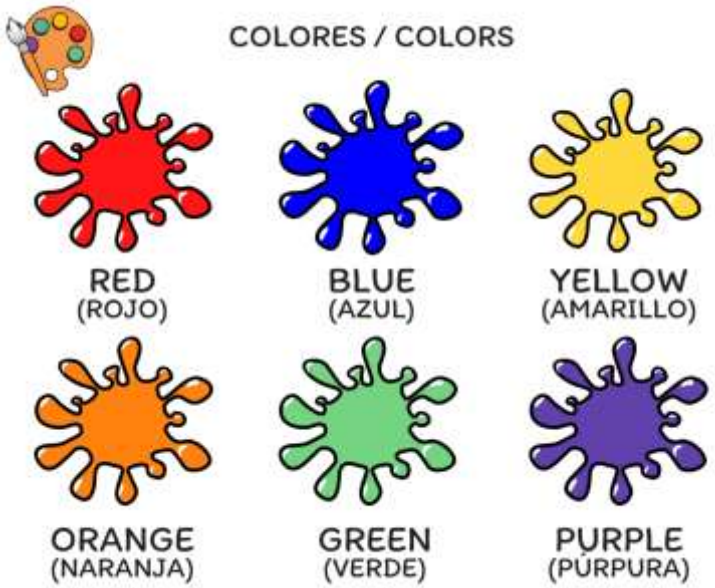
Estaciones del año

Crea una enumeración llamada "Estaciones" que tenga las cuatro estaciones del año como valores (Primavera, Verano, Otoño, Invierno). Luego, escriba un programa que permita al usuario ingresar una estación y muestre la estación siguiente en el ciclo de las estaciones. Por ejemplo, si el usuario ingresa "Verano", el programa debe mostrar "Otoño".

Ejercicios

Coleres

Crea una enumeración llamada "Colores" que tenga los colores básicos como valores (Rojo, Azul, Verde). Luego, escriba un programa que permita al usuario seleccionar un color y muestre su nombre en inglés y en español. Por ejemplo, si el usuario selecciona "Rojo", el programa debe mostrar "Red (Rojo)



Ejercicios

Lista de campeones

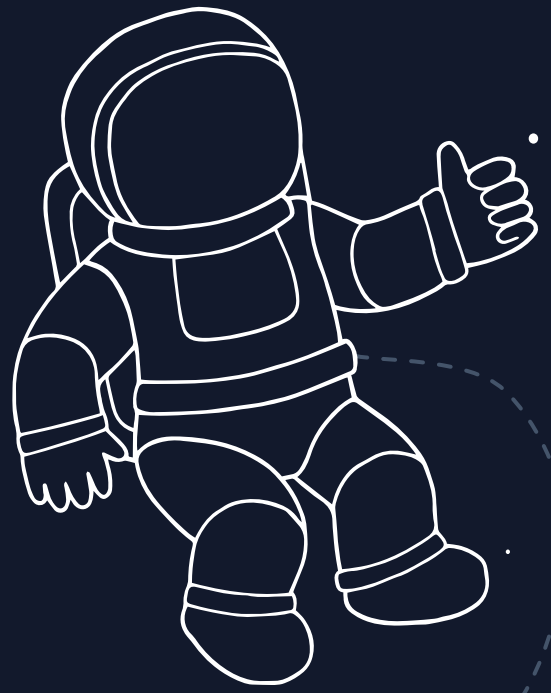
Crea un programa al que se le permita ingresar los nombre de los jugadores de equipos de futbol y en que ligas ha participado. Luego, el programa permitirá listar todos los jugadores que han participado en una liga en particular.

El listados de las ligas en las que puede participar un jugador son las siguiente:

Clasificación Mundial Sudamérica, Clasificación Copa África, Liga de las Naciones de la UEFA, Primera División, Segunda División, Liga de las Naciones CONCACAF, Copa del Rey, Champions League.

Gestione que para que solo permita ingresar las ligas permitidas y cualquier error otro que pueda darse.


OBJECT-ORIENTED
PROGRAMMING



Programa académico CAMPUS

Módulo JAVA