

The background is a light blue sky with several stylized autumn leaves in shades of orange, yellow, and green. On the left, there is a red brick barn with a yellow bell hanging from its roof. A yellow school bus is driving on a winding road that curves through a landscape of rolling green hills and stylized trees with orange and yellow foliage. In the bottom right corner, there are several large, orange pumpkins with green stems.

# Curso Java Intermedio

Christian Velázquez – Febrero 2017

<epam>

Bienvenue!



# En capítulos anteriores...

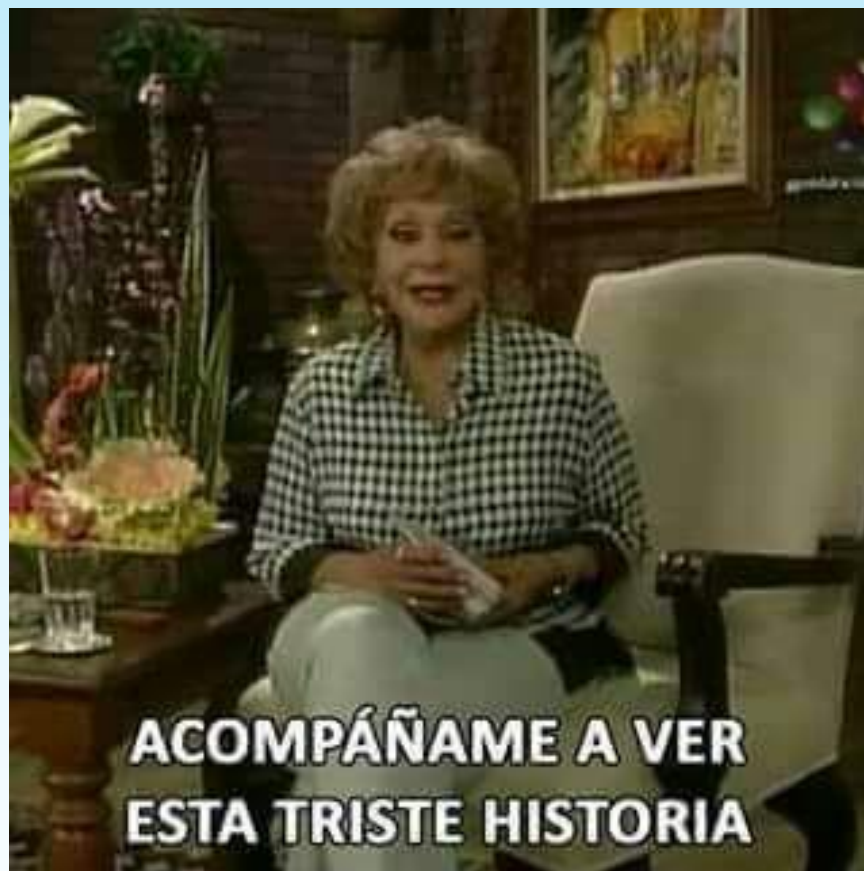
- Principios y conceptos de P00
- Modificadores, niveles de acceso e inicialización
- Variables
- Operadores
- Control de flujo
- Excepciones



# A continuación...





- **Colecciones**
- **Genéricos**
- **Multi-threading**

# A continuación...









# Qué son las colecciones?

- Son objetos que agrupan multiples elementos en una sólo unidad.
  - Se utilizan para trabajar con grupos de datos:
    - Almacenar
    - Obtener
    - Manipular
- 
- 
- 
- 



# Qué son las colecciones?

- Son objetos que agrupan multiples elementos en una sólo unidad.
  - Se utilizan para trabajar con grupos de datos:
    - Almacenar
    - Obtener
    - Manipular
- 
- 
- 
- 

# Qué son las colecciones?

- Para entender colecciones vamos a platicar de **estructuras de datos**. **Escoger la estructura de datos adecuada** para un problema **puede ser la diferencia** entre resolverlo óptimamente o no.
- En Java contamos con el **Java Collection Framework**.

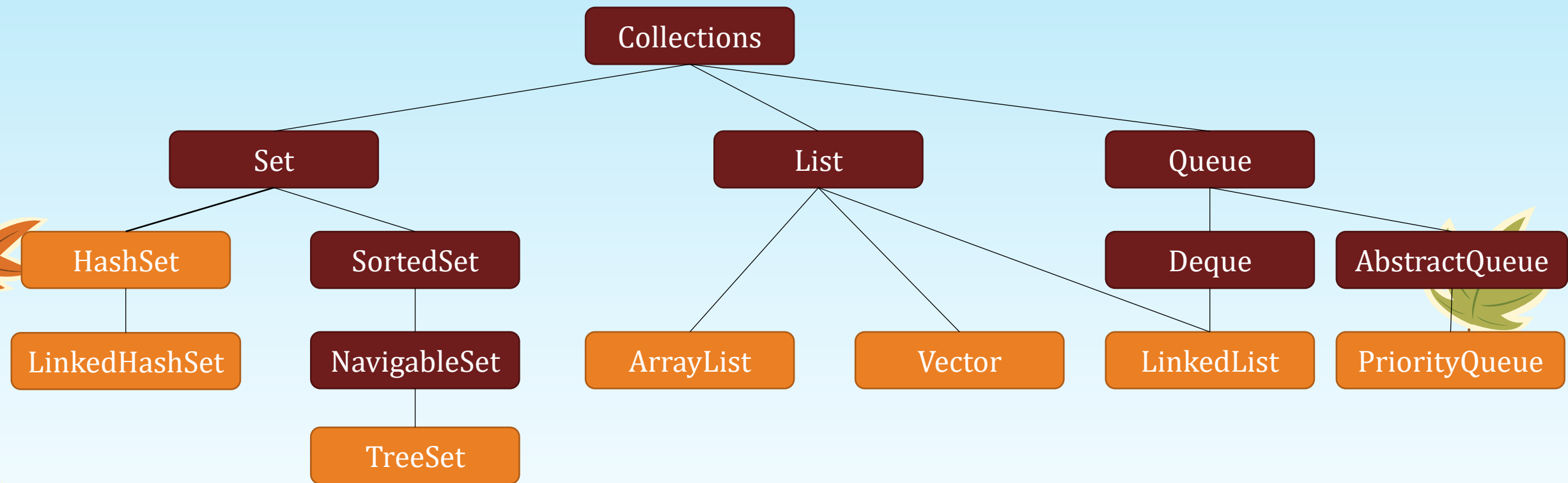


The slide features a light blue background with stylized autumn leaves in green, orange, and yellow scattered around the edges. At the bottom, there are rolling green hills. The title 'Java Collection Framework' is centered at the top in a bold, black, serif font.

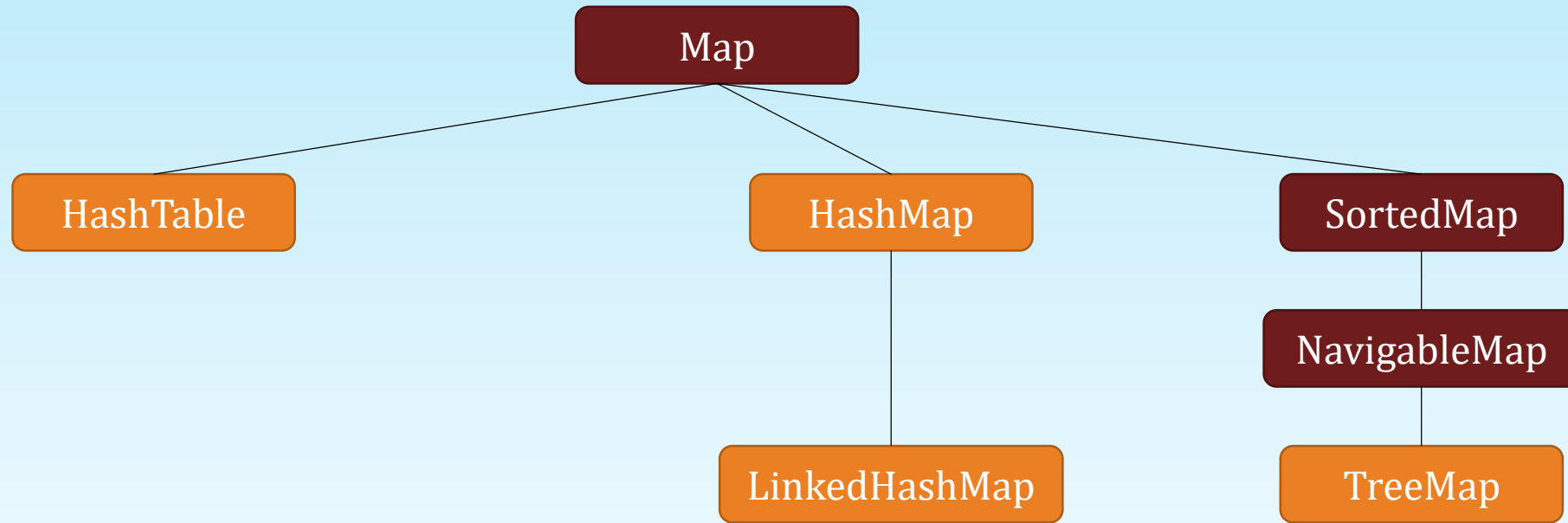
# Java Collection Framework


- Los *collection framework* se componen de:
  - Interfaces
  - Implementaciones
  - Algoritmos

# Jerarquía de colecciones

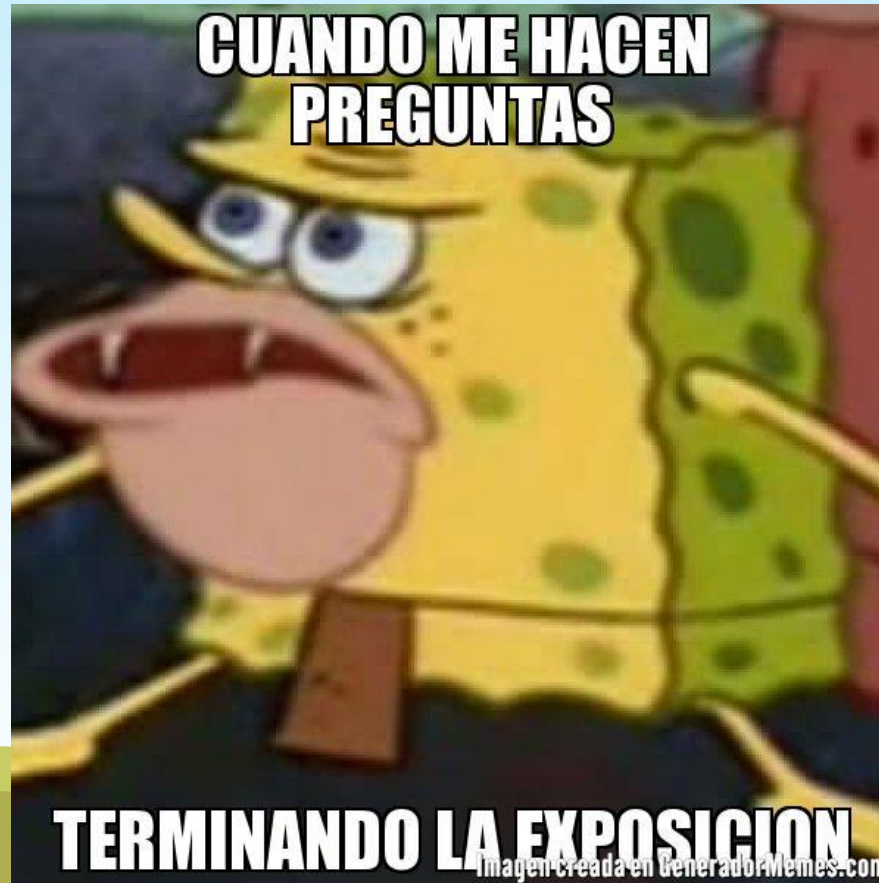


# Mapas







- 
- The slide features a light blue background with a green rolling hill at the bottom. Several stylized autumn leaves in shades of green, orange, and yellow are scattered around the edges. A list of methods is presented in a black sans-serif font.
- Todas las colecciones tienen métodos en común, por ejemplo:
    - add
    - addAll
    - contains
    - isEmpty
    - remove
    - removeAll
    - **iterator**

# Preguntas?









# Listas

- Es una estructura de datos básica en la cual, cada elemento de la lista tiene un *link* a otro dato del mismo tipo.
  - Permite implementar otro tipo de estructuras como pilas y colas.
  - **Veamos código...**
- 
- 
- 
- 





# Pilas y colas

- Las listas nos permiten implementar otras estructuras como bags, **pilas (stacks)** y **colas (queues)**.
  - **Los stacks son estructuras LIFO (Last In – First Out)**
  - **Las queues son estructuras FIFO (First In – First Out)**
- 
- 
- 
- 



# Pilas y colas



- En Java, las pilas y colas se pueden implementar utilizando un **LinkedList** u otras clases que implementen la interfaz **Deque**, como los **ArrayDeque**.
- 
- 







# Pilas y colas

- Existe también otro tipo de Queue conocido como **PriorityQueue**, que en lugar de ser **FIFO**, ordena los elementos bajo cierto criterio.
- 
- 



# Ejemplo



- Crear un método(s) que determine si una cadena de paréntesis está balanceada. Los caracteres válidos son `()[]{}`
- **Por ejemplo**
  - `(([]){})` es una cadena balanceada.
  - `{[]}()]` NO está balanceada

# Problema

- Crear un método que calcule el resultado de una expresión de operaciones de máximo dos operandos. Las expresiones estarán delimitadas por paréntesis ( ) y las operaciones válidas son las cuatro operaciones aritméticas básicas (+ - \* /)
- **Por ejemplo**
  - $(1 + (2 * 3)) = 7$
  - $(9 * (1 - (2 + 4))) = -45$





# Tipos Genéricos

- Los tipos genéricos (**Generics**) fueron agregados en Java 5 como una forma de solventar bugs en tiempo de compilación.
  - Permiten crear clases parametrizadas, es decir, clases que pueden ser re-utilizadas con más de un tipo de entrada.
- 
- 





# Tipos Genéricos





- El uso de Generics nos permite:
    - Chequeo de tipos en tiempo de compilación
    - Eliminar casting
    - Implementar algoritmos genéricos (que funcionen para más de un tipo).
- 
- 

# Tipos Genéricos

- El uso de Generics nos permite:
  - Chequeo de tipos en tiempo de compilación
  - Eliminar casting
  - Implementar algoritmos genéricos (que funcionen para más de un tipo).
- Las colecciones del Java Collection Framework están parametrizadas usando Generics, lo que nos permite utilizarlas con diferentes tipos de objetos.







# Set

- Los set son colecciones que no permiten tener elementos duplicados. Para determinar si un elemento ya se encuentra en la colección se utilizan los métodos **hashCode** y **equals**, heredados de la clase **Object**.
  - Nos enfocaremos en las implementaciones **HashSet**, **LinkedHashSet** y **TreeSet**.
- 
- 
- 
- 








# Map

- Los mapas son colecciones que almacenan datos en pares de **llave y valor**.
  - Podemos verlos como diccionarios, donde almacenamos un valor específico junto con una llave, que será utilizada para obtener el valor de vuelta.
  - Nos enfocaremos en las implementaciones **HashMap**, **LinkedHashMap** y **TreeMap**.
- 
- 
- 
- 









# Problema

- Tenemos un **arreglo de números** enteros y un número **num**. Encontrar todos los **pares** en el arreglo **cuya suma sea** igual a **num**.
  - Por ejemplo:
    - Arreglo = [1, 4, 7, 9, 6, 0, 3]
    - Num = 10
    - Pares = [{1,9}, {4,6}, {7,3}]
- 
- 
- 
- 
- 







# Map

- Los mapas son colecciones que almacenan datos en pares de **llave y valor**.
  - Podemos verlos como diccionarios, donde almacenamos un valor específico junto con una llave, que será utilizada para obtener el valor de vuelta.
  - Nos enfocaremos en las implementaciones **HashMap**, **LinkedHashMap** y **TreeMap**.
- 
- 
- 
- 



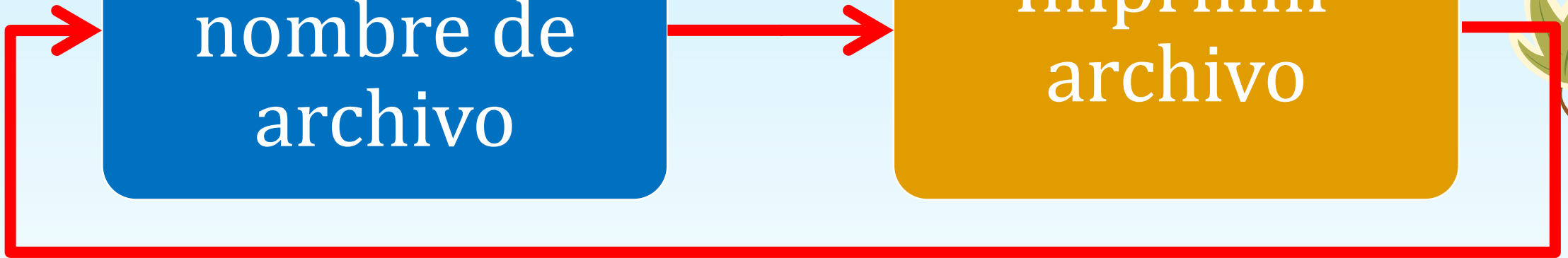
# Concurrencia

- Los hilos o **Threads** son un mecanismo que nos permite ejecutar tareas de manera paralela.
  - Existen casos donde un programa puede ser ejecutado “por partes” ya que se compone de varias tareas que no dependen necesariamente unas de las otras.
- 
- 
- 
- 

# Concurrencia

Solicitar  
nombre de  
archivo

Imprimir  
archivo







# Cómo crear un Thread en Java

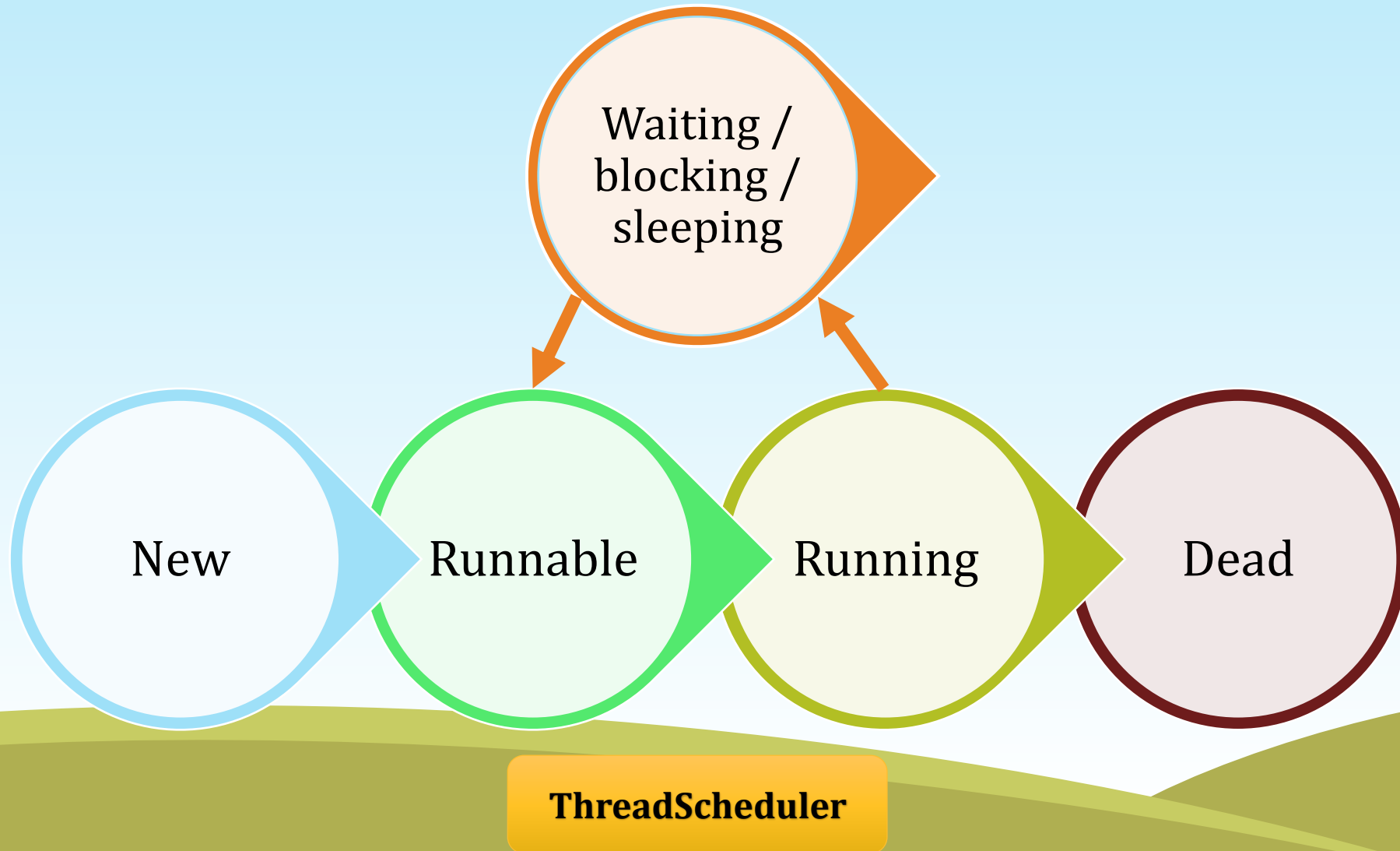
- Implementar la interfaz **java.lang.Runnable**
  - Implementar el método **run()**
- Extender la clase **java.lang.Thread**
  - Sobreescibir el método **run()**



# Cómo crear un Thread en Java

- En cualquier caso, se debe crear una instancia de la clase Thread utilizando nuestra implementación de Thread (aquel que implementa Runnable o extiende de Thread).
  - La creación y ejecución del hilo ocurre hasta que se llama el método **start()**
- 
- 
- 
- 

# Estados de los hilos



# Influencia sobre el ThreadScheduler

- sleep
  - yield
  - join
  - setPriority
- 
- wait
  - notify
  - notifyAll



# Preguntas?

