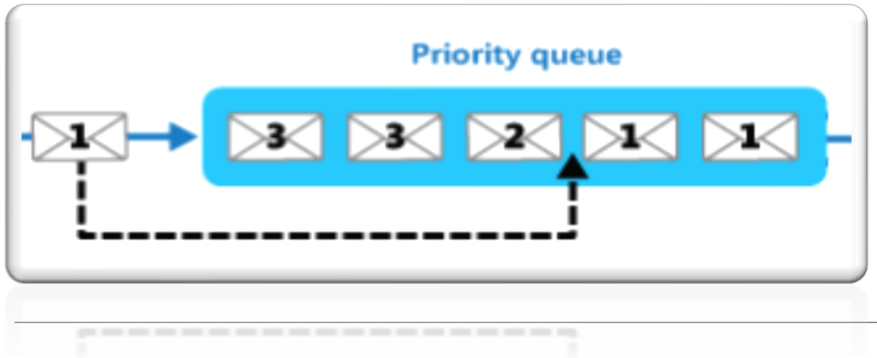
[Estructuras de Datos]



COLAS CON PRIORIDAD.

Copyright

COLAS CON PRIORIDAD

Introducción: ¿qué es un cola con prioridad?

- Una **cola con prioridad (CPP)** es un colección de elementos, llamados valores, los cuales tienen **asociada una prioridad** que es **provista** en el momento que el elemento es **insertado**.
- Las **prioridades** de cada elemento **insertado** establecen un **orden total**, a partir del cual los **valores** son **ordenados**.
- Operaciones fundamentales de una **CCP**.
  - **insertar(Clave,Valor):** Inserta un ***valor*** con prioridad ***clave*** en la **CCP**.
  - **removerMinimo():** Retorna y remueve de la **CCP** una **entrada** con la prioridad más pequeña.

Introducción: ¿qué es un cola con prioridad?

- Notar que tanto en la **inserción** como en la **eliminación**, se hace uso de la noción de **orden** entre las **prioridades**.
  - Las **entradas** de una **CCP** se ordenan en relación a sus **prioridades**.
  - Las **prioridades** de las **entradas** quedan establecidas por el **orden total** definido a partir de sus **claves**.
  - Notar que la **clave** más **pequeña** define la entrada con **mayor prioridad**.
- Por ejemplo, dadas dos entradas con claves y valores enteros *e1=<15,1560>* y *e2=<1,101>*, la entrada ***e2*** **tiene mayor prioridad** que ***e1***, justamente porque su clave asociada (1) **es menor** que la clave asociada a la entrada ***e1*** (15).
  - Ordenamiento de claves: 1 < 15 (léase k2 es menor que k1).
  - Ordenamiento de entradas: e1 < e2 (léase e2 **tiene prioridad** sobre e1).

CCP :: ED Subyacente

- Para que una **CCP** funcione adecuadamente es indispensable que exista un **comparador** de prioridades a fin de establecer la **relación de orden**.
  - Dadas dos **claves**: ¿son iguales? ¿Cuál es la menor? ¿Cuál es la mayor?
  - Tres alternativas para la implementación de una CCP.

```
public class CCP_con_alguna_ED <K extends Comparable<K>,V> implements PriorityQueue<K,V>{

  protected Comparator<K> comparador;
  // Otros atributos necesarios para la implementación.

  public CCP_con_alguna_ED (){
    comparador = new DefaultComparator<K>();
  }
  ...
}
```

*La ED que implementa la **CCP** puede hacer uso de un **comparador por defecto**. Para esto requiere que las claves genéricas sean **comparables**.*

CCP :: ED Subyacente

- Para que una **CCP** funcione adecuadamente es indispensable que exista un **comparador** de prioridades a fin de establecer la **relación de orden**.
  - Dadas dos **claves**: ¿son iguales? ¿Cuál es la menor? ¿Cuál es la mayor?
  - Tres alternativas para la implementación de una CCP.

```
public class CCP_con_alguna_ED <K,V> implements PriorityQueue<K,V>{

  protected Comparator<K> comparador;
  // Otros atributos necesarios para la implementación.

  public CCP_con_alguna_ED (Comparator<K> comp){
    comparador = comp;
  }
  ...
}
```

*La ED que implementa la **CCP** puede hacer uso de un **comparador** de claves que debe implementar el cliente, sin la necesidad de que las claves genéricas sean **comparables**.*

CCP :: ED Subyacente

- Para que una **CCP** funcione adecuadamente es indispensable que exista un **comparador** de prioridades a fin de establecer la **relación de orden**.
  - Dadas dos **claves**: ¿son iguales? ¿Cuál es la menor? ¿Cuál es la mayor?
  - Tres alternativas para la implementación de una CCP.

```
public class CCP_con_alguna_ED <K extends Comparable<K>,V> implements PriorityQueue<K,V>{

  protected Comparator<K> comparador;
  // Otros atributos necesarios para la implementación.

  public CCP_con_alguna_ED (){
    comparador = new DefaultComparator<K>();
  }

  public CCP_con_alguna_ED (Comparator<K> comp){
    comparador = comp;
  }
  ...
}
```

*La ED que implementa la **CCP** puede hacer uso de dos opciones: un **comparador por defecto** de claves, o un **comparador** que debe implementar el cliente. En ambos casos, se está requiriendo que las claves genéricas sean **comparables**.*

CCP :: Ejemplo de uso

- Asumir que se utilizará una **CCP** para realizar el **ordenamiento** de un conjunto de parciales.
- ¿**Cómo** utilizar la **CCP** para realizar tal tarea?
- ¿**Cuál** será la **prioridad** asociada a un **parcial**?
- Considere que se **ordenan** respecto a las **notas**.
  - ¿**Qué** sucede si **no parametrizo** un **comparador**?
  - ¿**Qué** nota será la de **mayor prioridad**?
  - ¿**Cuál** es el **ordenamiento** por **defecto**?
  - ¿**Cómo** se pueden **ordenar descendentemente**?

*Se puede definir una **CCP** cuyos **valores** son los **Parciales**, y cuyas **claves** sean datos **comparables** que indican la **prioridad** establecida para **ordenar** los **parciales**.*

*Dependerá del orden con el cual se desean ordenar. Podrían ordenarse por **Nombre**, por **Nota**, por **Fecha**, etc.*

Se ordenarán con el comparador de float por defecto (asumiendo notas como floats).

La menor nota de la CCP.

Notas ordenadas de menor a mayor.

Redefiniendo el comparador de forma tal que si dos notas n1 y n2 se ordenan naturalmente como n1 < n2, entonces el comparador las considera exactamente al revés, esto es, n2 < n1.

CCP :: Ejemplo de uso

- Asumir que se utilizará una **CCP** para realizar el **ordenamiento** de un conjunto de parciales respecto a las **notas**.

```
public class Tester {
  public static void main(String [] args) {
    // Cola con prioridad que considera con mayor prioridad los parciales con notas menores.
    PriorityQueue<Float, Parcial> ccp_asc = new CCP_con_alguna_ed<Float,Parcial>();

    // Cola con prioridad que considera con mayor prioridad los parciales con notas mayores.
    Comparator<Float> mi_comparador = new Comparador_por_defecto_inverso<Float>();
    PriorityQueue<Float, Parcial> ccp_des = new CCP_con_alguna_ed<Float,Parcial>(mi_comparador);
    ...
  }
}
```

```
public class Comparador_por_defecto_inverso <E extends Comparable<E>> implements Comparator<E> {
  @Override
  public int compare(E e1, E e2) {
    return (-1) * (e1.compareTo(e2));
  }
}
```

Fin de la presentación.