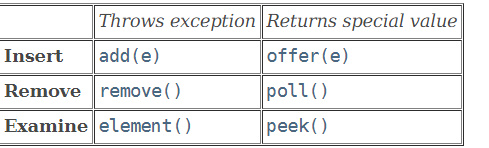


enqueue() – adaugarea itemelor

dequeue() – returnarea si stergerea itemelor

peek() – returnarea primului item fara a-l sterge.

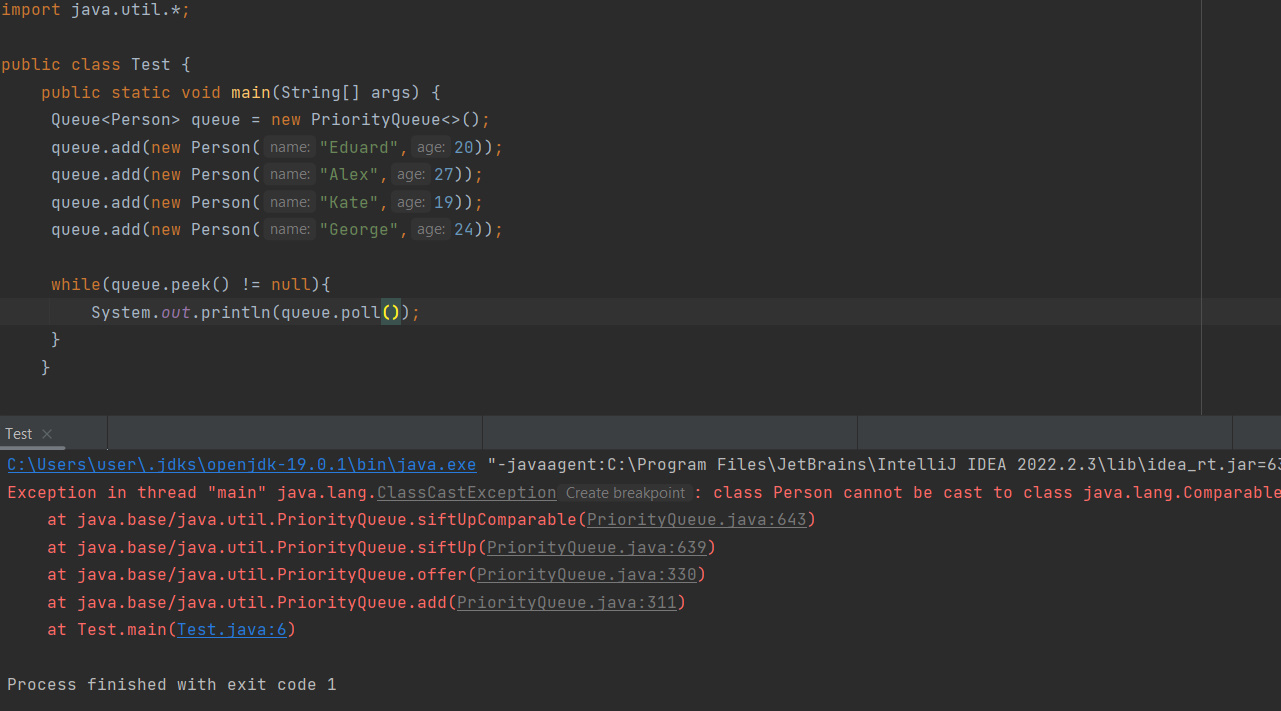
* Queue poate fi implementat si cu LinkedList si cu ArrayList, dar cu LinkedList e mai rapid si eficient, in general.
* LinkedList implementeaza Deque, si Deque extinde Queue
* Queue suporta operatii doar la Head, nu si Tail
* **Metode specifice doar lui Queue:**
* remove()– sterge primul element si il returneaza, adica Head. Daca nu sunt elemente, returneaza Exceptie
* poll() – sterge primul element si il returneaza. Returneaza null daca nu sunt elemente
* add() [**Collections interface]** – adauga un element. Daca nu poate, exceptia e aruncata, daca poate, returneaza true
* offer() – adauga un element. Daca nu poate, returneaza false, daca da, true
* element()[**Collections interface]** – returneaza head. Daca nu exista, arunca exceptie
* peek() – returneaza head. Daca nu exista, returneaza null



**Nu exista metoda get() caci nici Collection nu o are**

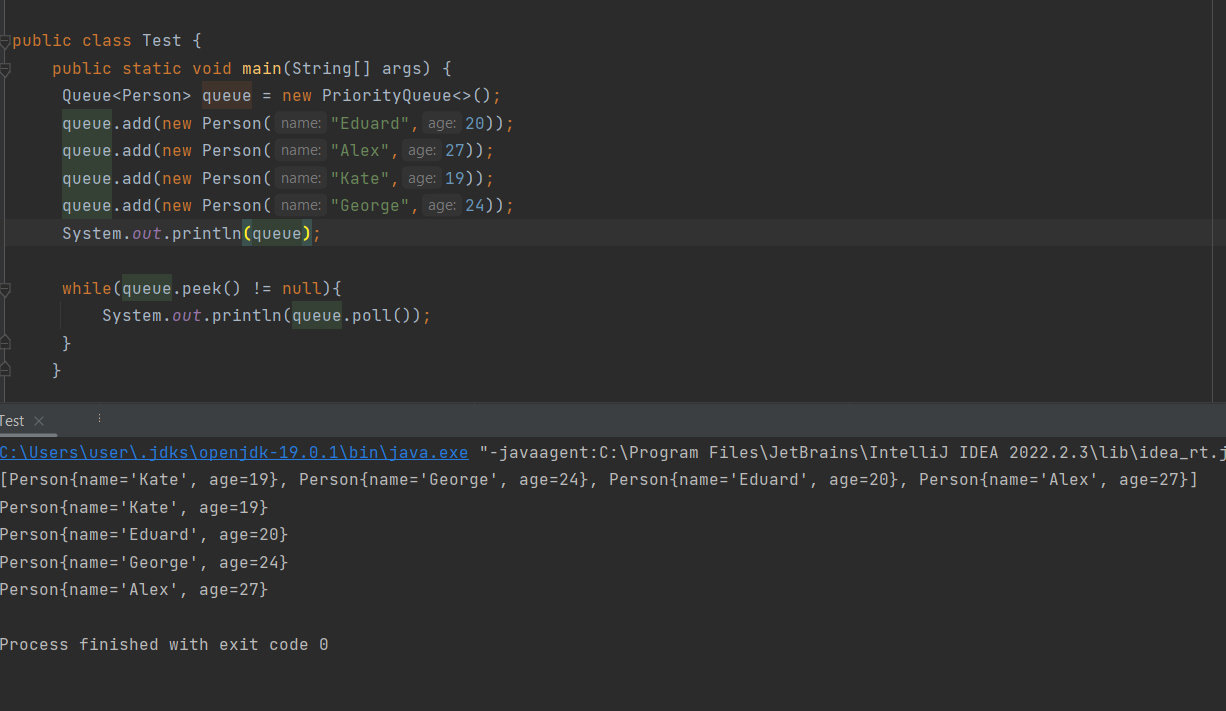
**PriorityQueue**

* Implementeaza Queue, nu Deque
* PriorityQueue se bazeaza pe aceea ca oferim o prioritate fiecarui element
* Un element cu o prioritate mai mare e servit inaintea unuia cu una mai mica
* Prioritatea se stabileste cu Comparable<> Interface sau Comparator<>
* **PriorityQueue e implementata cu array in java!**
* **Elementele care vor fi adaugate in PriorityQueue trebuie neaparat sa implementeze Comparable<> sau daca nu, trebuie neaparat sa dam un obiect ce implementeaza Comparator<> in constructor.** Daca nu, vom primi o eroare la run time, nu la compile time:





* Valorile null nu pot fi inserate
* PriorityQueue are referinta doar la Head
* PriorityQueue nu sorteaza toate elementele, dar se asigura ca elementul cu cea mai mare prioritate sa fie mereu primul. Cand acesta e sters sau un alt element e pus, se verifica ce element sa fie Head



Vedem ca doar Kate e in pozitia corecta, restul sunt puse cum nimereste. Apoi daca un element e sters, se cauta iar elementul ce trebuie pus in Head si daca se adauga altul, se verifica daca nu cumva el trebuie sa fie in Head

* **Constructors**:
* PriorityQueue() – creaza o pq cu size 11
* PriorityQueue(Collection c) – creaza o pq cu elementele din colectia data
* PriorityQueue(capacity) – creaza o pq cu capacitatea initiala
* PriorityQueue(Collection, Comparator) – creaza o pq cu elementele din collection si le sorteaza folosind comparatorul trimis. **Comparator, nu Comparable!**
* PriorityQueue(PriorityQueue) – creaza o pq cu elementele din alta pq
* PriorityQueue(SortedSet)
* PriorityQueue(Comparator)
* **Metode specifice:**
* **Atentie! Nu exista metoda get(), caci putem accesa doar primul item si atat si in plus nici Queue nici Collection nu o are**
* comparator() – returneaza comparatorul folosit

**Deque->ArrayDeque**

* Deque – double ended queue
* ArrayDeque Se creaza pe baza la array
* e pentru array foarte mari
* Deque Extinde Queue.
* ArrayDeque deci se bazeaza pe array ca ArrayList
* Avantajul sau e ca ne ofera referinta si la ultimul element si control asupra lui
* ArrayDeque este foarte rapid
* Datorita faptului ca putem manipula ultimul element, ArrayDeque poate fi folosit si ca un Stack
* ArrayDeque defapt si este un Stack imbunatatit
* E mai rapid ca Stack si ca LinkedList daca e folosit ca Queue
* **Constructors**:
* ArrayDeque() – creaza un ArrayDeque ce are 16 elemente default size
* ArrayDeque(Collection)
* ArrayDeque(capacity)
* **Deque/Queue Methods:**
* getFirst()
* getLast()
* offer(Item) – adauga la urma
* offerFirst(Item)
* offerLast(Item)
* poll() – returneaza Head si o sterge
* pop() – returneaza Head si o sterge
* push(Item)
* remove()
* removeFirst()
* removeFirstOccurrence(Item)
* removeLast()
* removeLastOccurrence(Item)
* peek() – returneaza Head
* peekFirst() – ca getFirs()
* peekLast() – ca getLast()
* pollFirst() – returneaza Head si o sterge
* pollLast() – returneaza Tail si o sterge