

1- Set Interface

- أهم ميزته فيه (no duplicated) أي كل العناصر اللي فيه تعتبر unique element
- الترتيب فيه ليست مهم => أي العناصر الذي داخله unsorted

1 some set method

| الوصف | الدالة |
|-----------------------------|--|
| Set تضيف عنصر إلى الـ | <code>add(E e)</code> |
| تضيف مجموعة من العناصر | <code>addAll(Collection<? extends E> c)</code> |
| يمسح كل العناصر | <code>clear()</code> |
| تتحقق هل العنصر موجود | <code>contains(Object o)</code> |
| تتحقق هل كل العناصر موجودة | <code>containsAll(Collection<?> c)</code> |
| آخر Set و Set مقارنة بين | <code>equals(Object o)</code> |
| فارغة Set تتحقق هل الـ | <code>isEmpty()</code> |
| للعناصر Iterator ترجع | <code>iterator()</code> |
| تزيل عنصر معين | <code>remove(Object o)</code> |
| تزيل مجموعة من العناصر | <code>removeAll(Collection<?> c)</code> |
| تحتفظ فقط بالعناصر المشتركة | <code>retainAll(Collection<?> c)</code> |
| ترجع عدد العناصر | <code>size()</code> |
| ترجع مصفوفة من العناصر | <code>toArray()</code> |

drawbacks

لا تسمح للوصول للعنصر by index

2-HashSet

HashSet set هو كلاس بيرث من ال

الخصائص الرئيسية 🧠

- ❌ لا يسمح بتكرار العناصر: أي محاولة لإضافة عنصر موجود بالفعل يتم تجاهلها.
- ❌ لا يحافظ على ترتيب العناصر: العناصر تُخزن بدون ترتيب محدد.
- ✅ يسمح بعنصر null واحد فقط.
- ✅ أداء عالي في العمليات الأساسية مثل: الإضافة – البحث – الحذف.
- ✅ يعتمد داخليًا على HashMap.

HashSet Methods

-نفس الميثود الموجوده في ال Set

متى نستخدم HashSet؟

- عندما نريد تخزين عناصر غير مكررة.
- عندما لا يهمنا ترتيب العناصر.
- عندما نحتاج إلى أداء سريع جدًا في العمليات الأساسية.

3 - LinkedHashSet => (set هو كلاس implement من ال linkedHashSet)

الخصائص الأساسية لـ LinkedHashSet

- لا يسمح بتكرار العناصر. ❌
- يحافظ على ترتيب الإدخال (insertion order). ✔️
- يسمح بإضافة عنصر **null** واحد فقط. ✔️
- الأداء جيد، لكن أبطأ قليلاً من **HashSet**. ✔️
- غير متزامن (Not Thread-Safe). ✔️
- يستخدم داخلياً **HashTable + LinkedList**. ➡️

متى نستخدم LinkedHashSet؟ 🔧

- عندما نحتاج إلى:
 - عناصر غير مكررة
 - مع الاحتفاظ بترتيب الإدخال
 - مثال: عرض بيانات المستخدمين حسب ترتيب دخولهم بدون تكرار.
-

مقارنة بين HashSet و LinkedHashSet





| الخاصية | HashSet | LinkedHashSet |
|--------------------|---------------|-------------------|
| يسمح بالتكرار؟ | ✗ | ✗ |
| يحافظ على الترتيب؟ | ✗ | ✓ |
| الأداء | أسرع | أبطأ قليلاً |
| يسمح بـ null؟ | ✓ (مرة واحدة) | ✓ (مرة واحدة) |
| الاستخدام | سرعة | ترتيب + عدم تكرار |

interface sortedSet (sortedSet extends from set)

ما هو SortedSet؟ 

SortedSet هو واجهة (Interface) فرعية من **Set** ضمن **Java Collection Framework**، تُستخدم لتخزين عناصر فريدة (non-duplicate) بترتيب تصاعدي تلقائي (natural ordering) أو حسب مقارن مخصص (Comparator).

الخصائص الرئيسية لـ SortedSet 

-  لا يسمح بتكرار العناصر.
-  يتم ترتيب العناصر تلقائياً (سواء طبيعي (تصاعدي) أو مخصص).
-  لا يسمح بإضافة أكثر من **null** إذا تم الترتيب الطبيعي.
-  يدعم الوصول إلى العناصر حسب الترتيب مثل أول وآخر عنصر.

متى تستخدم SortedSet؟

- عند الحاجة إلى:

- تخزين بيانات مرتبة.
- التأكد من عدم التكرار.
- الوصول إلى نطاق معين من العناصر (subset).

SortedSet في (Methods) أهم الدوال

| الوصف | الدالة |
|---|---|
| المستخدم للترتيب Comparator يرجع الـ | <code>comparator()</code> |
| (يرجع أول عنصر (الأصغر | <code>first()</code> |
| (يرجع آخر عنصر (الأكبر | <code>last()</code> |
| يرجع مجموعة من العناصر أقل من العنصر المحدد | <code>headSet(E toElement)</code> |
| يرجع مجموعة من العناصر أكبر أو تساوي العنصر | <code>tailSet(E fromElement)</code> |
| يرجع مجموعة فرعية بين عنصرين | <code>subSet(E fromElement, E toElement)</code> |
| يضيف عنصر | <code>add(E e)</code> |
| يزيل عنصر | <code>remove(Object o)</code> |
| يتحقق من وجود عنصر | <code>contains(Object o)</code> |

الفرق بين Set و SortedSet






| الخاصية | Set | SortedSet |
|--------------------|-----------------------------|----------------|
| يسمح بالتكرار؟ | ✗ | ✗ |
| يحافظ على الترتيب؟ | ✗ (إلا في LinkedHashSet) | ✓ |
| يسمح بالنطاقات؟ | ✗ | ✓ |
| الترتيب | غير محدد أو حسب الإدخال | تلقائي أو مخصص |

TreeSet (treeSet is class implement the sortedSet)

ما هو **TreeSet**? 📌

TreeSet هو كلاس في Java ينفذ الواجهتين **NavigableSet** و **SortedSet** ويُستخدم لتخزين عناصر غير مكررة مرتبة تلقائيًا.

الخصائص الأساسية لـ **TreeSet**

-  لا يسمح بتكرار العناصر.
 -  يحافظ على ترتيب العناصر تلقائيًا.
 -  لا يسمح بإضافة **null** إذا كانت المجموعة تحتوي على عناصر أخرى.
 -  يدعم العمليات على نطاقات من البيانات.
 -  يعتمد داخليًا على شجرة حمراء-سوداء (Red-Black Tree).
-

متى نستخدم TreeSet؟ 

عندما نحتاج إلى:

- تخزين بيانات مرتبة + غير مكررة

- التعامل مع نطاقات مثل:

- عناصر أكبر من X

- عناصر أقل من Y

- البحث السريع عن أول وآخر العناصر

أهم الدوال (Methods) في TreeSet

الوصف

الدالة

إضافة عنصر

`add(E e)`

إزالة عنصر

`remove(Object o)`

التحقق من وجود عنصر

`contains(Object o)`

يرجع أول عنصر

`first()`

يرجع آخر عنصر

`last()`

أول عنصر أكبر من المعطى

`higher(E e)`

أول عنصر أصغر من المعطى

`lower(E e)`

أول عنصر \leq العنصر المعطى

`ceiling(E e)`

أول عنصر \geq العنصر المعطى

`floor(E e)`

جميع العناصر أقل من قيمة معينة

`headSet(E toElement)`


جميع العناصر أكبر من أو تساوي قيمة

`tailSet(E fromElement)`

مجموعة من العناصر بين قيمتين

`subSet(E from, E to)`

مقارنة بين TreeSet و HashSet و LinkedHashSet

| LinkedHashSet | HashSet | TreeSet | الخاصية |
|---|---|---|--------------------|
|  |  |  | يسمح بالتكرار؟ |
|  (ترتيب الإدخال) |  |  (ترتيب طبيعي أو مخصص) | يحافظ على الترتيب؟ |
|  (مرة واحدة) |  (مرة واحدة) |  | يسمح بـ null؟ |
| متوسط | أسرع ($O(1)$) | أبطأ ($O(\log n)$) | الاداء |
|  |  |  | يدعم النطاقات؟ |

ملاحظات هامة

- لا يُفضل استخدام **TreeSet** مع عناصر غير قابلة للمقارنة (non-comparable) بدون توفير **Comparator**.
- لو حاولت إضافة **null** بجانب عناصر أخرى → ستحصل على **NullPointerException**.
- عند المقارنة بين الكائنات، يجب أن تكون من نفس النوع المتوافق (مثلاً لا تخلط بين **String** و **Integer**).

PriorityQueue(PriorityQueue implement Queue)

ما هي PriorityQueue؟ 

PriorityQueue هي كلاس في Java تنفذ واجهة **Queue**، وتستخدم لتنفيذ بنية بيانات تشبه الطابور، ولكن:

العناصر تُزال حسب الأولوية، وليس حسب الترتيب الذي أُضيفت به.

◆ تقع ضمن الحزمة:

المفهوم الأساسي

● **PriorityQueue** ترتب العناصر داخليًا حسب الأولوية (بشكل افتراضي: تصاعديًا – من الأصغر إلى الأكبر).

● تستخدم عادة heap داخلي.

● العنصر ذو الأولوية الأعلى هو أول من يُزال.

متى نستخدم PriorityQueue؟

- عندما تريد التعامل مع عناصر حسب الأولوية.

- أمثلة:

- جدولة المهام حسب الأهمية.

- أنظمة المعالجة الفورية.

- الخوارزميات مثل Dijkstra.

ملاحظات مهمة

- لا يسمح بالقيم **null**.

- غير متزامن (Not Thread-Safe)، لكن يمكن جعله متزامنًا باستخدام:

PriorityQueue لا تضمن ترتيب العناصر عند الطباعة **(toString)**، لأنها مرتبة داخليًا في هيكل **heap**.

مقارنة مع Queue العادية

| الخاصية | Queue | PriorityQueue |
|---------------|----------------------|-----------------------------|
| ترتيب العناصر | حسب الإدخال (FIFO) | حسب الأولوية |
| يدعم null؟ | نعم (في بعض الأنواع) | ❌ |
| طباعة العناصر | تظهر بالترتيب | لا تعكس الترتيب الحقيقي |
| استخدام شائع | الطابور العادي | أنظمة الأولوية، الخوارزميات |

الخلاصة

- **PriorityQueue** مناسبة عندما تحتاج إلى:
 - إزالة العنصر ذو الأولوية الأعلى أولاً.
 - ترتيب تلقائي حسب قيمة العنصر أو باستخدام **Comparator**.
- تُستخدم في:
 - تنفيذ أنظمة جدولة المهام.

○ الخوارزميات التي تعتمد على الترتيب مثل Dijkstra و *A.

● مرنة جدًا ويمكن تخصيصها لأي نوع بيانات.

Deque (Deque extend from Queue)

📌 ما هو **Deque**؟

Deque هي اختصار لـ **Double Ended Queue**، وهي واجهة في Java تسمح بإضافة وحذف العناصر من:

● البداية (front)

● النهاية (rear)

أي إنها تدعم:

● **Stack (FILO)** ↔ باستخدام نهاية واحدة فقط.

● **Queue (FIFO)** ↔ باستخدام الطرفين.

🧠 المفهوم الأساسي

● **Deque** هي طابور ثنائي النهاية (double-ended queue).

● يمكن استخدامه كـ **Stack** أو **Queue**.

● أهم تطبيق عملي له في Java هو باستخدام **ArrayDeque**.

متى نستخدم Deque؟ 🔧

● عندما نحتاج إلى:

- إدخال أو حذف عناصر من الطرفين.
- محاكاة (Stack (LIFO أو (Queue (FIFO.
- أداء أفضل من Stack و LinkedList.

ملاحظات مهمة

- ArrayDeque هو التطبيق الأكثر كفاءة من Deque.
- لا يدعم العناصر null.
- أسرع من LinkedList عند الاستخدام كـ stack أو queue.
- غير متزامن (Not thread-safe).

مقارنة بين Queue و Deque

Deque

Queue

الخاصية

من الطرفين

من طرف واحد

الإدخال والحذف



يدعم Stack؟



يدعم Queue؟

ArrayDeque, LinkedList

LinkedList, PriorityQueue

مثال تطبيقي

أهم الدوال في Deque

الوصف

الدالة

يضيف عنصر في البداية

`(addFirst(E e`

يضيف عنصر في النهاية

`(addLast(E e`

يضيف في البداية (بدون استثناء عند الخطأ)

`(offerFirst(E e`

يضيف في النهاية

`(offerLast(E e`

يزيل أول عنصر

`(removeFirst`

يزيل آخر عنصر

`(removeLast`

يزيل ويُرجع أول عنصر أو `null`

`(pollFirst`

يزيل ويُرجع آخر عنصر أو `null`

`(pollLast`

يعرض أول عنصر

`(peekFirst`

يعرض آخر عنصر

`(peekLast`

الخلاصة

- **Deque** واجهة مرنة تسمح بالتعامل مع البيانات من الطرفين.
- يمكن استخدامها ك:
 - Stack (LIFO) باستخدام **push()** و **pop()**.
 - Queue (FIFO) باستخدام **offer()** و **poll()**.
- استخدام **ArrayDeque** مع **Deque** يعطي أداء عالي وفعال.
- مهم جدًا في البرمجة التنافسية وخوارزميات مثل:

Sliding Window ○

Palindrome Checker ○

Undo/Redo history ○






● ArrayDeque(ArrayDeque implement Deque)

ما هو **ArrayDeque**? 

ArrayDeque هو كلاس (class) في Java يُستخدم لتنفيذ واجهة **Deque** ((Double-Ended Queue)). يسمح بإضافة أو إزالة العناصر من كلا الطرفين (البداية والنهاية).

يتم تخزين العناصر داخليًا باستخدام مصفوفة ديناميكية (Resizable Array).

المميزات الأساسية لـ ArrayDeque

-  أسرع من **LinkedList** عند استخدامه كـ **Stack** أو **Queue**.
-  لا يسمح بالقيم **null**.
-  غير متزامن (Not Thread-safe).
-  يتمدد تلقائيًا إذا امتلأت المصفوفة.
-  يُستخدم بديلًا عن **Stack** و **LinkedList** في الكثير من الحالات.

متى نستخدم ArrayDeque؟

- عند الحاجة إلى:
 - **Stack (LIFO)** باستخدام **push()** و **pop()**.
 - **Queue (FIFO)** باستخدام **offer()** و **poll()**.
 - أداء أعلى من **LinkedList**.
 - عمليات سريعة من الطرفين.

ملاحظات مهمة

- لا يُسمح بالقيم **null** لأن هذا قد يسبب تعارضًا مع دوال مثل **poll()**.
- الأداء أسرع بكثير من **Stack** و **LinkedList** من حيث:
 - الإضافة.

○ الإزالة.

○ الوصول للعناصر.

○