## الشريحة 1

**Chapter 5: The JAVA Collections Framework**
الفصل الخامس: إطار عمل المجموعات في جافا

عندما تحتاج إلى تنظيم عدة كائنات داخل برنامجك، يمكنك وضعها داخل مجموعة (Collection).
يتم تنفيذه بواسطة **فئة أو أكثر (classes)** كل نوع من **الواجهات (interface)**.
تجمع المجموعة العناصر معًا وتتيح الوصول إليها واسترجاعها لاحقًا.

---

## 🟦 الشريحة 2

**Collections Framework Diagram**
مخطط إطار عمل المجموعات

كل فئة من فئات المجموعات تنفذ واجهة من **تسلسل هرمي (hierarchy)**.
تم تصميم كل فئة لتناسب نوعًا محددًا من التخزين.

---

## 🟦 الشريحة 3

**Collection Interface**
واجهة المجموعة

**Each collection class implements an interface from a hierarchy**
**Each class is designed for a specific type of storage**

🔹

- في لغة **(Collections Framework)** في إطار عمل **المجموعات (class)** كل **فئة** من **(hierarchy)** تنتمي إلى **تسلسل هرمي (interface)** تقوم **بتنفيذ واجهة** Java الواجهات.
  وكل فئة مثل ... `Collection`, `List`, `Set`, `Map` 🔸 أي أن هناك واجهات أساسية مثل `ArrayList`, `HashSet`, `TreeMap` تطبق إحدى هذه الواجهات.
- كل **فئة** تم تصميمها لغرض **تخزين نوع معين من البيانات** أو لتناسب **طريقة معينة في التنظيم والوصول** إلى العناصر.
  مثلًا 🔸 :
  - فئة **ArrayList** مناسبة عندما تحتاج **الوصول السريع** إلى العناصر حسب الفهرس.
  - فئة **LinkedList** مناسبة عندما تحتاج **إدخال أو حذف العناصر بسرعة** من أي موضع.

- مناسبة عندما تحتاج إلى **عناصر فريدة فقط** وبحث سريع جدًا **HashSet** فئة.

🟢 **ببساطة:**
تطبق واجهة معينة (لتحديد سلوكها العام)، **Collections Framework** كل فئة في الـ
وهي مصممة لتخدم **نوعًا محددًا من التخزين أو الاستخدام** في البرنامج.

---

## 🟦 الشريحة 4

**List Interface**
واجهة القائمة

القائمة هي مجموعة **تحتفظ بترتيب عناصرها**.
يوجد فئتان تنفذان هذه الواجهة:

- **ArrayList**: يخزن العناصر في مصفوفة قابلة لتغيير الحجم ديناميكيًا.
- **LinkedList**: يسمح **بإضافة وإزالة العناصر بسرعة** من القائمة.

---

## 🟦 الشريحة 5

**Set Interface**
واجهة المجموعة غير المرتبة

المجموعة (Set) هي مجموعة **غير مرتبة** من **العناصر الفريدة**.
ترتب عناصرها بطريقة تجعل **البحث، الإضافة، والإزالة** أكثر كفاءة.

فئتان تنفذانها:

- **HashSet**: يستخدم **جداول التجزئة (hash tables)** لتسريع العمليات.
- **TreeSet**: يستخدم **شجرة ثنائية (binary tree)** لتسريع البحث والإضافة والإزالة.

---

## 🟦 الشريحة 6

**Stacks and Queues**
المكدسات والطوابير

طريقة أخرى لزيادة الكفاءة هي **تقليل عدد العمليات المتاحة**.
مثالان على ذلك:

- **Stack (المكدس)**: يحتفظ بترتيب العناصر، لكنه لا يسمح بإدخال عناصر في أي موضع، يمكن فقط **إضافة وإزالة العناصر من الأعلى**.
- **Queue (الطابور)**: تضيف العناصر من **نهاية واحدة (الذيل)**، وتزيلها من **النهاية الأخرى (الرأس)**.
  مثال: طابور من الناس ينتظرون عند موظف بنك.

---

## 🟦 الشريحة 7

**Maps Interface**
واجهة الخرائط

الخريطة (Map) تخزن **المفاتيح (keys)**، و**القيم (values)**، و**العلاقات بينهما**.
مثال: رموز الباركود كمفاتيح، والكتب كقيم.

- **المفاتيح (Keys):** وسيلة سهلة لتمثيل كائن (مثل رقم باركود أو رقم هوية طالب).
- **القيم (Values):** الكائن الفعلي المرتبط بالمفتاح.

الخريطة تحافظ على **العلاقة بين المفتاح والقيمة**.

---

## 🟦 الشريحة 8

**The Collection Interface (1)**
واجهة المجموعة (الجزء 1)

**Collection** هي **واجهات متخصصة** ترث من واجهة **Set** و**Queue** و**List** الواجهات.
**الأساليب (methods)** المستخدمة بشكل شائع تشترك جميعها في مجموعة من.

---

## 🟦 الشريحة 9

**The Collection Interface (2)**
واجهة المجموعة (الجزء 2)

(تكملة للأساليب والخصائص المشتركة)

---

## 🟦 الشريحة 10

**Linked Lists Class**
فئة القوائم المترابطة

القوائم المترابطة تستخدم **المراجع** **(references)** للحفاظ على قائمة مرتبة من **العُقد** **(nodes)**.

- الرأس (head): يشير إلى أول عقدة.
- كل عقدة تحتوي على **قيمة** **(value)** و**مرجع للعقدة التالية**.

يمكن استخدامها لتنفيذ:

- واجهة **List**
- واجهة **Queue**

---

## 🟦 الشريحة 11

**Linked Lists Operations**
عمليات القوائم المترابطة

- **عمليات فعالة:**
  - **الإدراج:** تحديد الموضع الصحيح وتحديث المراجع.
  - **الإزالة:** تحديد العنصر المراد حذفه وتحديث مراجع الجيران.
  - **زيارة جميع العناصر بالترتيب.**
- **عمليات غير فعالة:**
  - **الوصول العشوائي (Random Access)**

كل متغير مثيل يُعلن تمامًا مثل أي متغير آخر استخدمناه سابقًا.

---

## 🟦 الشريحة 12

**LinkedList: Important Methods**
الأساليب المهمة في LinkedList

(توضيح لطرق الإضافة، الإزالة، والبحث في القائمة)

---

## 🟦 الشريحة 13

**List Iterators Interface**
واجهة مكررات القوائم (ListIterator)

**ListIterator** الذي يحتفظ بموقعك في القائمة. استخدم **LinkedList**، عند **المرور عبر** يمكن استخدامه لـ:

- الوصول إلى العناصر داخل القائمة.
- زيارة العقد التي ليست الأولى أو الأخيرة.

مثال في جافا:

```
LinkedList<String> employeeNames = ...;
ListIterator<String> iter = employeeNames.listIterator();
```

---

## 🟦 الشريحة 14

**Using Iterators**
استخدام المكررات

فكر في المكرر (iterator) كأنه **مؤشر بين عنصرين** (مثل المؤشر في محرر النصوص).
يجب أن يكون **النوع العام** **(generic type)** للمكرر مطابقًا لنوع القائمة.

```
iterator.next();
iterator.add("J");
ListIterator<String> iter = myList.listIterator();
```

---

## 🟦 الشريحة 15

**Iterator and ListIterator Methods**
أساليب Iterator و ListIterator

تتيح المكررات التحرك عبر القائمة بسهولة — مشابهة لمتغير الفهرس في المصفوفة.

---

## 🟦 الشريحة 16

**Iterators and Loops**
المكررات والحلقات

يُستخدم المكرر غالبًا داخل **حلقات** **while** و**for-each**.

- `hasNext()` تُرجع true إذا وُجد عنصر تالٍ.
- `next()` تُرجع قيمة العنصر التالي.

مثال:

```
while (iterator.hasNext()) {
    String name = iterator.next();
    // تنفيذ عملية
}

for (String name : employeeNames) {
```

```
    // تنفيذ عملية
}
```

---

## 🟦 الشريحة 17

**ListDemo.java (1)**
يُوضح كيفية **إضافة العناصر، إزالتها، وطباعة القائمة**.

---

## 🟦 الشريحة 18

**ListDemo.java (2)**
(تكملة لبرنامج العرض العملي على القوائم).