المحاضرة الثالثة
العودية

Abdalmohaymen alesmaeel

abdmoh87@gmail.com

محاور الجلسة

- ماهي التوابع العودية
- بنية التوابع العودية
- مثال عن التوابع العودية
- مقارنة الأداء بين التوابع العودية والحلقات
- الأخطاء الشائعة

الدوال العودية

- الدوال العودية هي نوع من الدوال في البرمجة، حيث تستدعي الدالة نفسها داخل تعريفها. ويتم استخدامها عادةً لحل المشاكل التي تتضمن تكرار عملية ما على نفس البيانات.
- تعمل الدوال العودية بتقسيم المشكلة إلى مشاكل صغيرة، ويتم حل كل مشكلة صغيرة باستخدام نفس الدالة العودية. وعندما يتم حل جميع المشاكل الصغيرة، يتم إعادة تجميع النتائج لإنتاج الحل النهائي للمشكلة الأصلية.
- يمكن تقديم التعريف السابق عن طريق مثال بسيط. فمثلاً، إذا كان لدينا مشكلة حساب عاملي لعدد معين، يمكننا استخدام دالة عودية لحل هذه المشكلة. ويتم ذلك عن طريق تقسيم المشكلة إلى مشاكل صغيرة، حيث يتم حساب عامل التجميع للعدد الأصغر بمقدار واحد. ويتم استخدام نفس الدالة العودية لحساب العاملي للعدد الأصغر، وذلك حتى يتم الوصول إلى العدد 1، ويتم حساب العامل التجميع له بسهولة.
- وبما أن الدالة تستدعي نفسها داخل تعريفها، فإنه يجب تحديد شرط يتوقف عنده الاستدعاء العودي ويتم إرجاع النتيجة المطلوبة. في حالة عاملي عدد، يكون هذا الشرط عندما يتم الوصول إلى العدد 1

بنية التابع العودية

```
public static void main(String[] args) {
  int f=factorial(10);
  System.out.println(f);
}
public static int factorial(int n){
    if (n==1) {
        return 1;
    }
    return n*factorial(n-1);
}
```

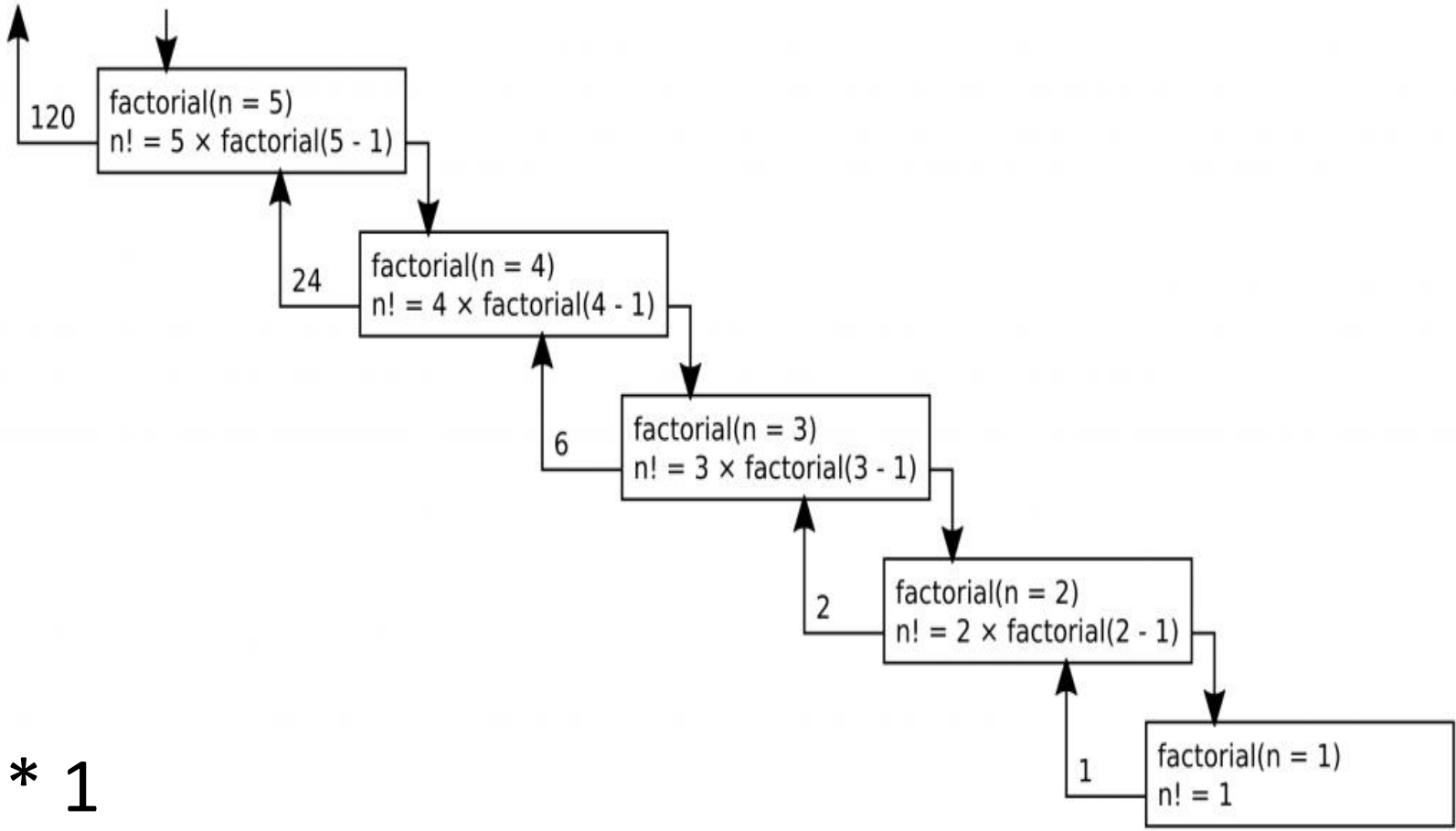
كتلة التابع العودية

شرط التوقف

الاستدعاء العودي

استدعاءات التابع العودي

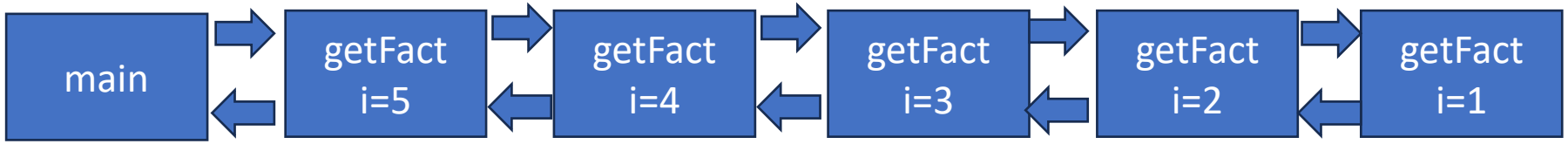
```
factorial(5)
5 * factorial(4)
5 * 4 * factorial(3)
5 * 4 * 3 * factorial(2)
5 * 4 * 3 * 2 * factorial(1)
5 * 4 * 3 * 2 * 1
120
```

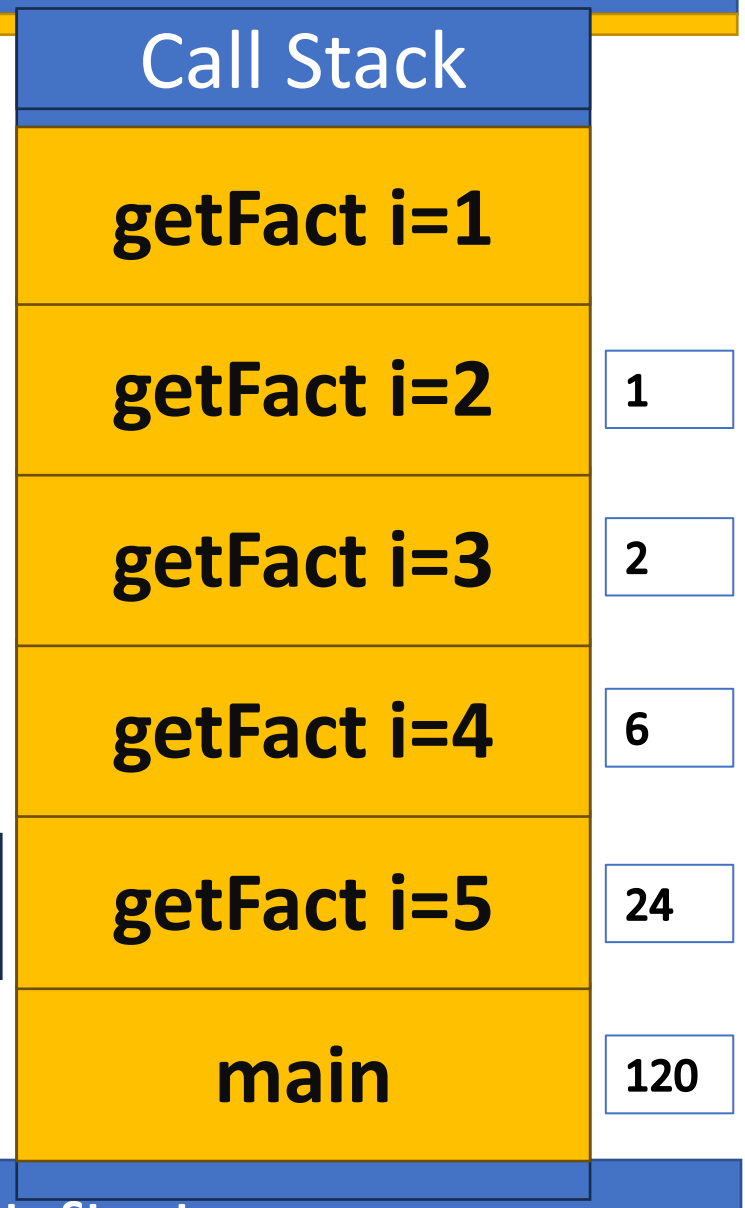


$$n! = n * (n-1) * (n-2) * \ldots * 1$$

مكدّس الإستدعاءات Call Stack

```
public static void main(String[] args) {
    int r=getFact(5);
    System.out.println(r);
}
public static int getFact(int i){
    if (i==1)
        return 1;
    return i*getFact(i-1);
}
```

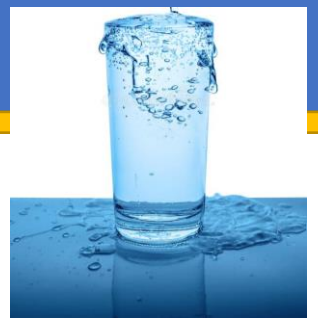




cs	cs	cs	cs					
				1				
			2	2	2			
		3	3	3	3	3 r=2		
	4	4	4	4	4	4	4 r=6	
5	5	5	5	5	5	5	5	5 24

Main=120

طفحان المكدس stack overflow



- **طفحان المكدس Stack Overflow :** هو مصطلح برمجي يشير إلى خطأ يحدث عندما يتجاوز البرنامج الحد الأقصى لحجم المكدس الذي يمكنه استخدامه.
- **المكدس في نظام التشغيل:** هو منطقة من الذاكرة تستخدم لتخزين المتغيرات المحلية وسجلات الاستدعاء للدوال، وهو مهم جدًا لتنفيذ الدوال وإدارة تدفق البرنامج.

يحدث طفحان المكدس عادةً بسبب:

- **الاستدعاء الذاتي المفرط أو الغير محدود للدالة (الاستدعاء الذاتي اللانهائي):** عندما تقوم دالة بالاتصال بنفسها مرارًا وتكرارًا دون شرط توقف واضح، مما يؤدي إلى استنفاذ المساحة المخصصة للمكدس.
- **استخدام كميات كبيرة جدًا من البيانات المحلية:** مثل تعريف مصفوفات كبيرة جدًا كمتغيرات محلية داخل الدوال.
- عندما يحدث طفحان المكدس يتوقف البرنامج عن العمل وقد يعطي رسالة خطأ توضح أن المكدس قد طُفح. هذا يمكن أن يكون صعب الكشف والتصحيح خاصةً في البرامج المعقدة.

لتجنب طفحان المكدس ينبغي:

- استخدام طرق التكرار بدلًا من **الاستدعاء الذاتي** المفرط حيثما كان ذلك ممكنًا.
- التأكد من وجود شروط توقف واضحة في الاستدعاءات الذاتية لضمان عدم الوقوع في حلقات لانهائية.
- تجنب تخزين كميات كبيرة من البيانات كمتغيرات محلية داخل الدوال.
- فهم كيفية عمل المكدس وإدارة الذاكرة في البرامج يمكن أن يساعد في كتابة كود أكثر كفاءة وتجنب أخطاء مثل طفحان المكدس.

```
Exception in thread "main" java.lang.StackOverflowError
        at recursion.Recursion.getFact(Recursion.java:9)
        at recursion.Recursion.getFact(Recursion.java:9)
```

الفرق بين الحلقات والاستدعاءات العودية

- تُستخدم الحلقات والتوابع العودية في تنفيذ نفس العملية مرارًا وتكرارًا، ولكن يختلفان في طريقة تنفيذهما واستخدامهما لموارد الذاكرة.
- **الحلقات:**
- تعتمد الحلقات على تنفيذ نفس العملية عدة مرات دون الحاجة إلى تحميل دالة أو طرد الدالة من الذاكرة بعد الانتهاء من تنفيذها. وبما أنها لا تحتاج إلى تخزين المتغيرات التي يتم انشاؤها ضمن الدالة في الذاكرة عدة مرات، فإنها تستخدم أقل موارد الذاكرة من التوابع العودية. ومع ذلك، يمكن أن يتم استخدام الحلقات في تنفيذ مهام أقل من المهام التي يمكن حلها باستخدام التوابع العودية، وليست مناسبة دائمًا لحل جميع المشاكل.
- **التوابع العودية:**
- تستخدم التوابع العودية لتنفيذ نفس العملية بشكل متكرر داخل الدالة نفسها، حتى يتم تحقيق الشرط الذي تم وضعه. ولأنها تستخدم نفس الدالة بشكل متكرر، فإنها تستخدم مزيدًا من موارد الذاكرة لتخزين متغيرات الاستدعاءات المتعددة للدالة في الذاكرة. ومع ذلك، يمكن استخدام التوابع العودية في حل المشاكل التي تتطلب حلول معقدة أو متغيرة، مما يجعلها أحيانًا أكثر فعالية من الحلقات.

الفرق بين الحلقات والاستدعاءات العودية

	التوابع العودية	الحلقات
توفير موارد الذاكرة	يتم حجز متغيرات لكل استدعاء	المتغيرات تبقى ثابتة ولا يتم هدر في حجز أماكن لمتغيرات اضافية
حل المشاكل المعقدة	جيد في حل المشاكل المعقدة	غير جيدة في المشاكل المعقدة

حساب العاملي باستخدام التوابع العودية

```
public static int factorial(int n){
    if (n==1) {
        return 1;
    }
    return n*factorial(n-1);
}
```

حساب العاملي باستخدام الحلقات

```
int fact=1;
for (int i = 10; i >0; i--) {
    fact*=i;
}
System.out.println(fact);
```

تنبيهات وأخطاء شائعة

- تجاوز حدود الذاكرة: يجب التأكد من عدم تجاوز حدود الذاكرة المخصصة للتوابع العودية، وذلك عن طريق التحقق من مقدار الذاكرة اللازمة لحفظ المتغيرات والقيم المؤقتة.
- تكرار الحلقات بشكل لا نهائي: يجب التأكد من إنهاء التوابع العودية بشكل صحيح لتجنب تكرار الحلقات بشكل لا نهائي وتجميد البرنامج.

```
public static int factorial(int n){
    return n*factorial(n-1);
}
```

- عدم اختبار الحالات الحدودية: يجب اختبار التوابع العودية في جميع الحالات الممكنة، بما في ذلك الحالات الحدودية مثل حالات القيم الصفرية والسالبة والكبيرة جدًا.
- الاستدعاء الذاتي غير الضروري: يجب تجنب الاستدعاء الذاتي غير الضروري للتوابع العودية، حيث يمكن أن يؤدي ذلك إلى زيادة الوقت اللازم لتنفيذ التوابع وزيادة استخدام الذاكرة.

تنبيهات وأخطاء شائعة

- عدم التحقق من المدخلات: يجب التحقق من المدخلات في التوابع العودية قبل استخدامها، وذلك لتجنب وجود مدخلات غير صالحة والتي قد تؤدي إلى حدوث أخطاء في البرنامج

- استخدام الذاكرة بشكل غير فعال: يجب تجنب استخدام الذاكرة بشكل غير فعال في التوابع العودية، حيث يمكن استخدام المؤشرات والمتغيرات المؤقتة لتخزين القيم بدلاً من استخدام متغيرات جديدة في كل دورة من التوابع.

- الاعتماد على التوابع العودية بشكل كبير: يجب تجنب الاعتماد على التوابع العودية بشكل كبير في البرنامج، حيث يمكن أن يؤدي ذلك إلى زيادة استخدام الذاكرة والوقت المطلوب لتنفيذ البرنامج.

- عدم توثيق التوابع العودية: يجب توثيق التوابع العودية بشكل جيد، وذلك لتسهيل قراءة وفهم الكود من قبل المبرمجين الآخرين، ولتجنب الأخطاء المحتملة في المستقبل.

امثلة-طباعة الاعداد من رقم

- طباعة الاعداد من رقم الى الرقم 1
- احدى الجوانب السلبية في الكود انه اذا تم ادخال رقم اصغر من صفر فانه سيتابع الى اللانهاية

```
public static void countUp(int n) {
    if (n==1) {
        System.out.println(1);
        return;
    }
    System.out.println(n);
    countUp(n-1);
}
```

امثلة – حساب جداء عددان

```
public static int multiply(int n, int m) {
    if (m == 0) {
        return 0;
    }
    return n + multiply(n, m - 1);
}
```

في الكود الجانبي يتم حساب جداء عددان حيث يتم مضاعفة عدد من خلال عدد آخر

امثلة تابع عودي يختبر اذا كان العنصر موجود ام لا

```
public static boolean isExist(ArrayList list,int index, int item)  {
    if (index==list.size()) {
        return false;
    }else if((int)list.get(index)==item){
        return true;
    }else{
        return isExist(list, index+1, item);
    }
}
```

تابع يقوم بالبحث عن محرف ضمن سلسلة نصية

```
public static boolean search(char target, String str) {
    if (str == null || str.length() == 0) {
        return false;
    } else if (str.charAt(0) == target) {
        return true;
    } else {
        return search(target, str.substring(1));
    }
}
```

تابع يقوم بحساب عدد تكرار محرف ضمن نص

```
public static int countOccurrences(char target, String str) {
    if (str == null || str.length() == 0) {
        return 0;
    } else if (str.charAt(0) == target) {
        return 1 + countOccurrences(target, str.substring(1));
    } else {
        return countOccurrences(target, str.substring(1));
    }
}
```

تابع لعكس سلسلة نصية

```
public static String reverseString(String str) {
    if (str.isEmpty()) {
        return str;
    } else {
        return reverseString(str.substring(1)) + str.charAt(0);
    }
}
```

تابع يقوم بحساب عدد المحارف ضمن نص

```
public static int countChars(String str) {
    if (str.isEmpty()) {
        return 0;
    } else {
        return 1 + countChars(str.substring(1));
    }
}
```

تابع يقوم بحذف محرف ضمن سلسلة نصية

```
public static String removeChars(String str, char c) {
    if (str.isEmpty()) {
        return str;
    } else if (str.charAt(0) == c) {
        return removeChars(str.substring(1), c);
    } else {
        return str.charAt(0) + removeChars(str.substring(1), c);
    }
}
```

أمثلة- توليد جمل عشوائية من النصوص

نلاحظ في الكود الجانبي انه تم وضع المتغيرات بشكل عام حتى لا يتم تخزينها عند كل استدعاء

```
private static Random random = new Random();
private static String[] nouns = {"cat", "dog", "tree", "house", "book"};
private static String[] verbs = {"runs", "jumps", "sings", "reads", "eats"};
private static String[] adjectives = {"happy", "big", "green", "loud", "clever"};
private static String[] articles = {"the", "a"};
public static String generateSentence() {
    String article = getRandomWord(articles);
    String noun = getRandomWord(nouns);
    String verb = getRandomWord(verbs);
    String adjective = getRandomWord(adjectives);
    if (random.nextInt(2) == 0) {
        return article + " " + noun + " " + verb + " " + adjective;
    } else {
        return article + " " + adjective + " " + noun + " " + verb + " " + generateSentence();
    }
}
public static String getRandomWord(String[] words) {
    return words[random.nextInt(words.length)];
}
```

```
public static void main(String[] args) {
        String sentence = generateSentence();
        System.out.println(sentence);
    }
```