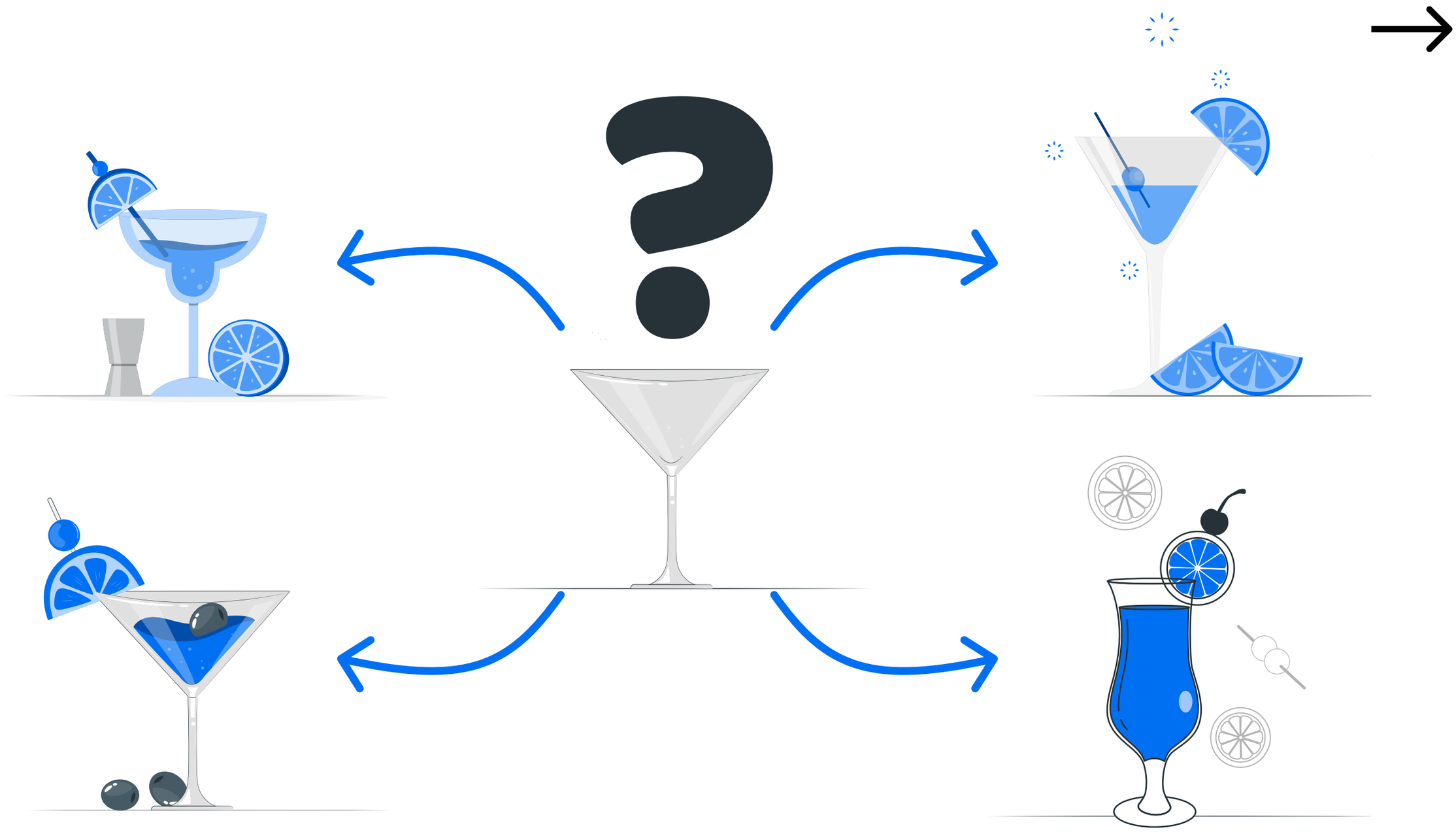
ط

ما هو الـ Dynamic Dispatch
وما علاقته بالـ Polymorphism ؟

ما معنى Dynamic Dispatch من الأساس ؟
وكيف نستخدمه بشكل عملي لجعل الكود اكثر مرونة وسلاسة ؟

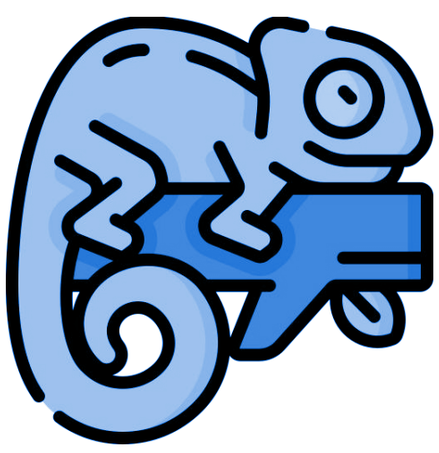


Ahmed El-Tabarani
Back-End Developer

تعريف الـ Polymorphism

مفهوم الـ Polymorphism يمثل قدرة أي شيء سواء دالة او object على أخذ عدة أشكال في آن واحد في مواقف مختلفة

أي أنها تتصرف بشكل مختلف كتغير لنوعها وصفتها أو تغير الـ implementation الخاصة بها بناءًا على حالة معينة



أمثلة على هذا هو الـ Overloading أو الـ Overriding، أو مع الـ object في حالة الـ Dynamic Dispatch

Ahmed El-Tabarani
Back-End Developer

أنواع الـ Polymorphism

ينقسم إلى نوعين

Runtime

قدرة اللغة او البرنامج على التعرف أو ادراك الـ Polymorphism
بعد تنفيذ كودك وأثناء ما البرنامج قيد التشغيل، وأنواعه:

- **Function Overriding**
- **Dynamic Dispatch**

- الـ Function Overriding هي عملية إعادة بناء لدالة موجودة في الـ Parent أي الكلاس الأساسي وتريد أن تعدلها في الكلاس الذي سيرثه

لاحظ كيف قام الـ Student بتغير الـ Implementation

```ts
class Person {
  public getInfo(){
    return "I am a person";
  }
}

class Student extends Person {
  public getInfo(){
    return "I am a student";
  }
}

let student = new Student()
student.getInfo() // I am a student
```

- الـ Dynamic Dispatch هو ما سوف نستفيض فيه باقي الشرح إن شاء الله

Compiler-Time

قدرة اللغة او البرنامج على التعرف أو ادراك الـ Polymorphism
اثناء كتابتك للكود وقبل تنفيذ البرنامج، وأنواعه:

- **Function Overloading**
- **Operator Overloading**
- **Generic/Template**

- الـ Function Overloding هي مجموعة دوال تشترك في نفس الإسم وتختلف في عدد الـ Parameters ونوعها
ونفس الأمر ينطبق على الـ Operator Overloding

لاحظ نفس الاسم
لكن parameters مختلف

```ts
function add(x: number, y: number) {
  return x + y;
}
function add(x: number, y: number, z: number) {
  return x + y + z;
}
function add(x: string, y: string) {
  return x + y;
}
add(5, 10); // 15
add(5, 10, 15); // 30
add('Ahmed', ' El-Tabarani'); // Ahmed El-Tabarani
```

- الـ Generic هي دوال وكلاسات تشترك في الـ implementation لكن تختلف فقط في نوع Datatype الذي يتم تحديده

```ts
function add<T>(x: T, y: T) {
  return x + y;
}

add<number>(5, 10); // 15
add<string>('Ahmed', ' El-Tabarani'); // Ahmed El-Tabarani
```

حدد نوع الـ datatype هنا

Ahmed El-Tabarani
Back-End Developer

ما هو الـ Dynamic Dispatch ؟

هو إمكانية التغير بشكل سلس ومرن في أي وقت
مثل الـ object يمكنه أن يغير نوع
الـ constructor الذي يتم إنشاؤه منه في أي وقت

فمثلًا لدينا عم أيمن من نوع إنسان، ويمكنه أن يكون
موظف عندما يدخل الشركة، و رب الأسرة عندما
يدخل بيته، وصيادٌ في البحر وهكذا ...

فلدينا كيان يمكنه أن يتخذ
صفات وأنواع مختلفة
في حالات معينة

Ahmed El-Tabarani
Back-End Developer

لنأخذ مثالًا عمليًا

لدينا كلاس يدعى Employee ولدينا أيضًا كلاسين CareemEmployee وSutraEmployee يرثان من الـ Employee

في مثالنا جعلنا الـ Employee كلاس عادي
لكن يمكنك أن تجعله abstract class أو interface
وسيتم تطبيق الـ dynamic dispatch دون مشاكل

```
class Employee {
  public getInfo() {
    return 'I am an employee';
  }
}

class CareemEmployee extends Employee {
  public getInfo() {
    return 'I am a Careem employee';
  }
}

class SutraEmployee extends Employee {
  public getInfo() {
    return 'I am a Sutra employee';
  }
}
```

كل كلاس قام بعمل overriding لدالة getInfo بشكل اعتيادي

Ahmed El-Tabarani
Back-End Developer

الآن لدينا كلاس Employee تحته مجموعة من الكلاسات التي ترث منه، هنا يمكن لمتغير من نوع Employee أن يكون object من نوع

SutraEmployee أو CareemEmployee

يمكن للـ employee أن يستدعي أي constructor يريده

```
let employee : Employee;
```

```
new CareemEmployee();
```

```
new SutraEmployee();
```

لكن يجب أن تكون الكلاسات ترث أو تبني الـ Employee
لكي يستطيع المتغير employee أن يستدعي أي constructor يريده

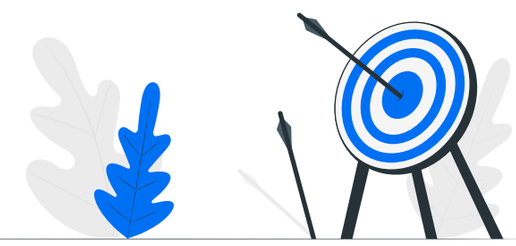
```
let employee: Employee;

employee = new CareemEmployee();
employee.getInfo(); // I am a Careem employee

employee = new SutraEmployee();
employee.getInfo(); // I am a Sutra employee
```

لاحظ كيف للمتغير من نوع Employee
أن أصبح object من نوع CareemEmployee
واستدعى دالة getInfo الخاص به

وثم أصبح object من نوع SutraEmployee
واستدعى دالة getInfo الخاص به أيضًا



هذا هو تحديدًا الـ Dynamic Dispatch
وهو التغير السلس للـ object بأن يصبح نوع مختلف في أي
وقت يريده فكان مرة موظف عند كريم وكان موظف عند سُترة

Ahmed El-Tabarani
Back-End Developer

لنأخذ مثالًا أخر لكي تتضح الصورة أكثر

لدينا abstract class يدعى Bank به دالة تمثل السحب ودالة تحضر كمية المال

وكل كلاس يرثه عليه أن يعيد بناء دالة withdraw

جعلناه abstract class هذه المرة
ويمكنك أن تجعله interface مع مراعاة الاختلاف

```ts
abstract class Bank {
  constructor(protected amount: number) {}

  public abstract withdraw(amount: number): void;
  public getAmount() {
    return `Bank amount ${this.amount}`;
  }
}
class MisrBank extends Bank {
  public withdraw(amount: number) {
    let amountWithTax = amount + amount * 0.1;
    this.amount -= amountWithTax;
  }
}
class CarioBank extends Bank {
  public withdraw(amount: number) {
    let amountWithTax = amount + amount * 0.5;
    this.amount -= amountWithTax;
  }
}
```

كل كلاس قام بعمل overriding لدالة withdraw وأخذ الضريبة الخاصة بكل بنك

Ahmed El-Tabarani
Back-End Developer

سننشيء دالة مستقلة ليست لها علاقة بأي كلاس

فقط دالة بسيطة اعتيادية، لكن المميز فيها أنها ستستقبل

أي object من نوع Bank

```
function logBankAmount(bank: Bank) {
  console.log(bank.getAmount());
}
```

لكن الـ Bank هو حاليًا abstract class وممكن أن يكون

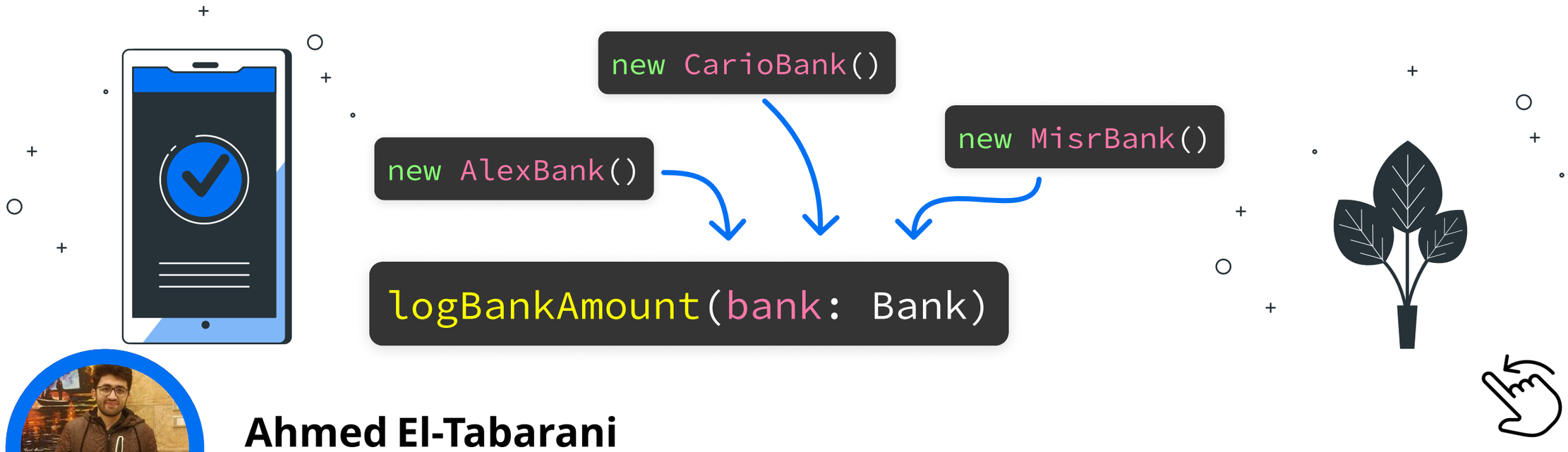
interface، بالتالي لن يكون له object

لكن بما أننا عرفنا مفهوم dynamic dispatch

الخاص بالـ polymorphism وكيف نستعمله ونستفيد منه

فنحن نعلم أن المتغير bank الذي تستقبله الدالة يمكنه أن

يتخذ أي constructor من الكلاسات التي تندرج تحت Bank



Ahmed El-Tabarani
Back-End Developer

المثال بسيط نعرف متغير من نوع Bank ثم نجعله يكون object من نوع MisrBank ثم قمنا ببعض الحسابات ثم غيرنا نوع البنك

```
let bank: Bank;

bank = new MisrBank(10000);
bank.withdraw(1000);

logBankAmount(bank); // Bank amount 8900

// Change the bank from Misr To Cario
bank = new CarioBank(10000);
bank.withdraw(1000);

logBankAmount(bank); // Bank amount 8500
```

الأمر المثير للاهتمام هي دالة logBankAmount التي استقبلت MisrBank في المرة الأولى ثم CarioBank في المرة الثانية بسلاسة دون مشاكل

هذا بسبب أنها تستقبل أي object تندرج تحت Bank

```
function logBankAmount(bank: Bank) {
  console.log(bank.getAmount());
}
```

لاحظ أن الدالة لا تهتم بنوع البنك، هي فقط تستقبل أي بنك من نوع Bank وتستدعي دالة getAmount التي ستتواجد في كل البنوك لانها معرفة في داخل الـ Bank

Ahmed El-Tabarani
Back-End Developer

أنا أحمد الطبراني، مهندس برمجيات SWE 😁

مبرمج متخصص في عالم الـ Backend ⚙️💻

أحب دائمًا أن أُشارك معرفتي المتواضعة مع الآخرين لعله عسى أن يستفيد شخص ما بما أكتبه 🤗
أكتب بعض المقالات عن أشياء مختلفة في عالم البرمجة، أرجوا أن تستفيدوا وتستمتعوا 😉

تابعني علي لينكدإن او علي مدونتي الشخصية