BACKEND

SEP 30, 2022

SOLID Principles

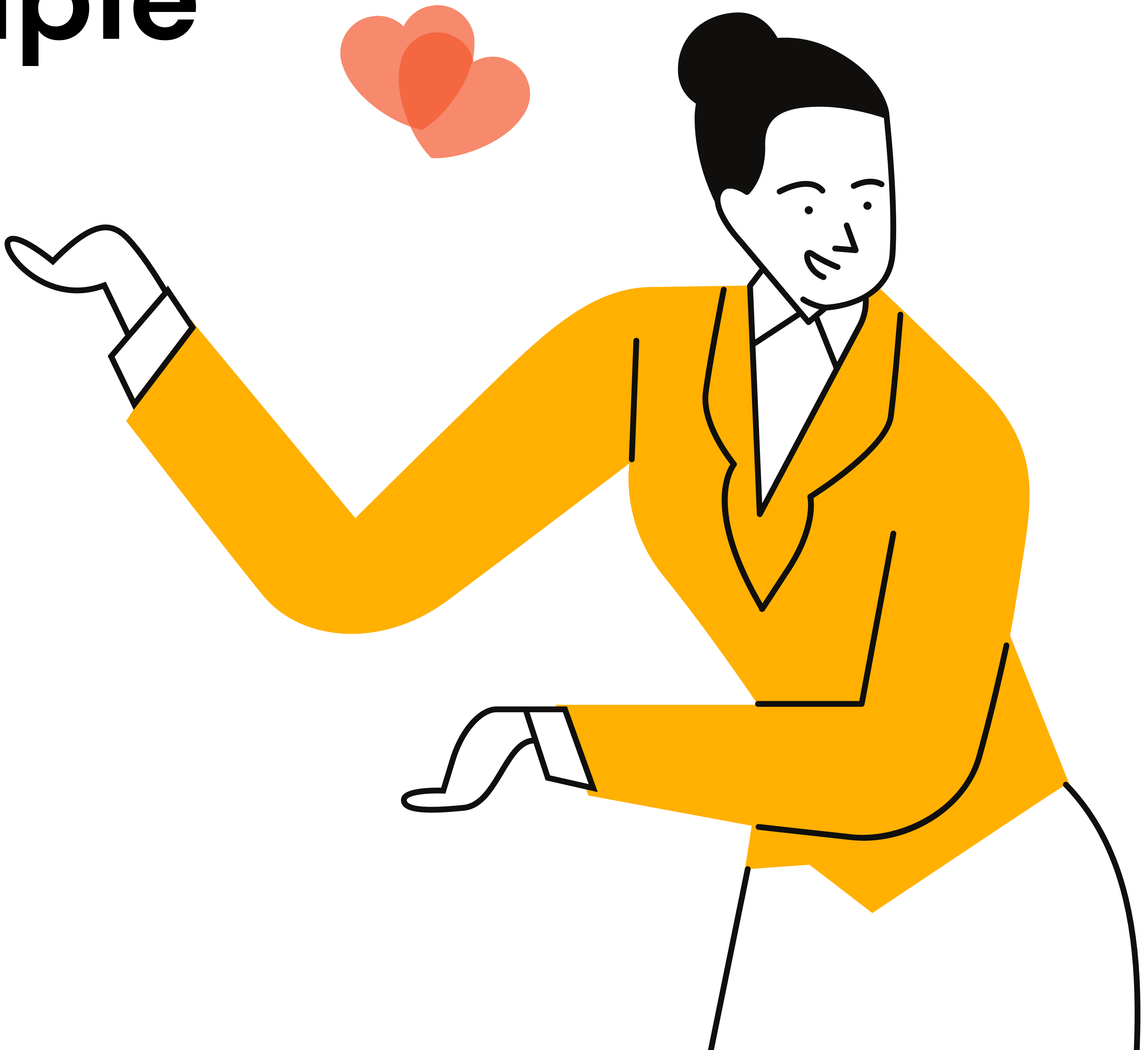
Diego Morales
Esteban Castaño
Jose Martinez

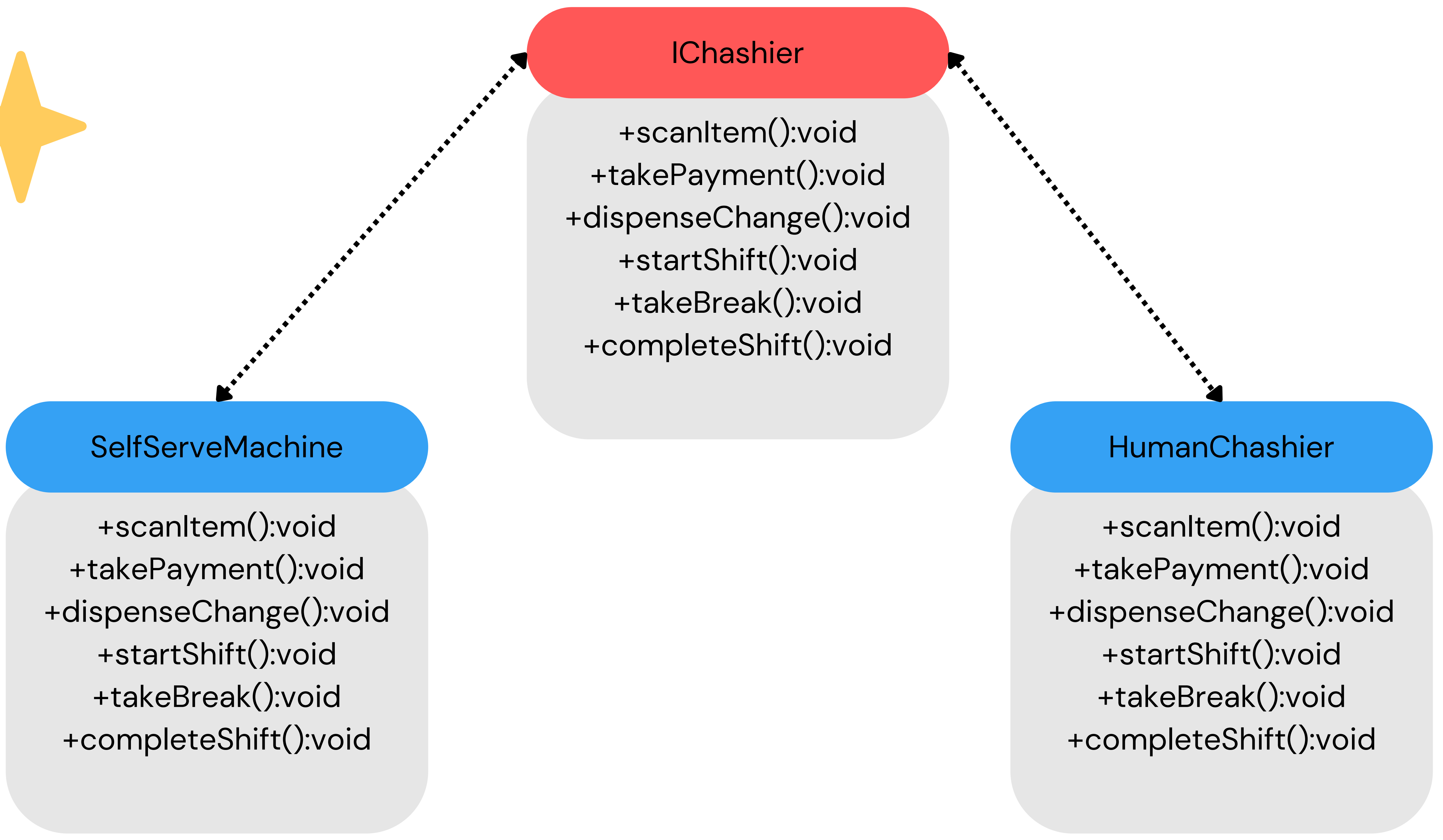
Back to Agenda Page

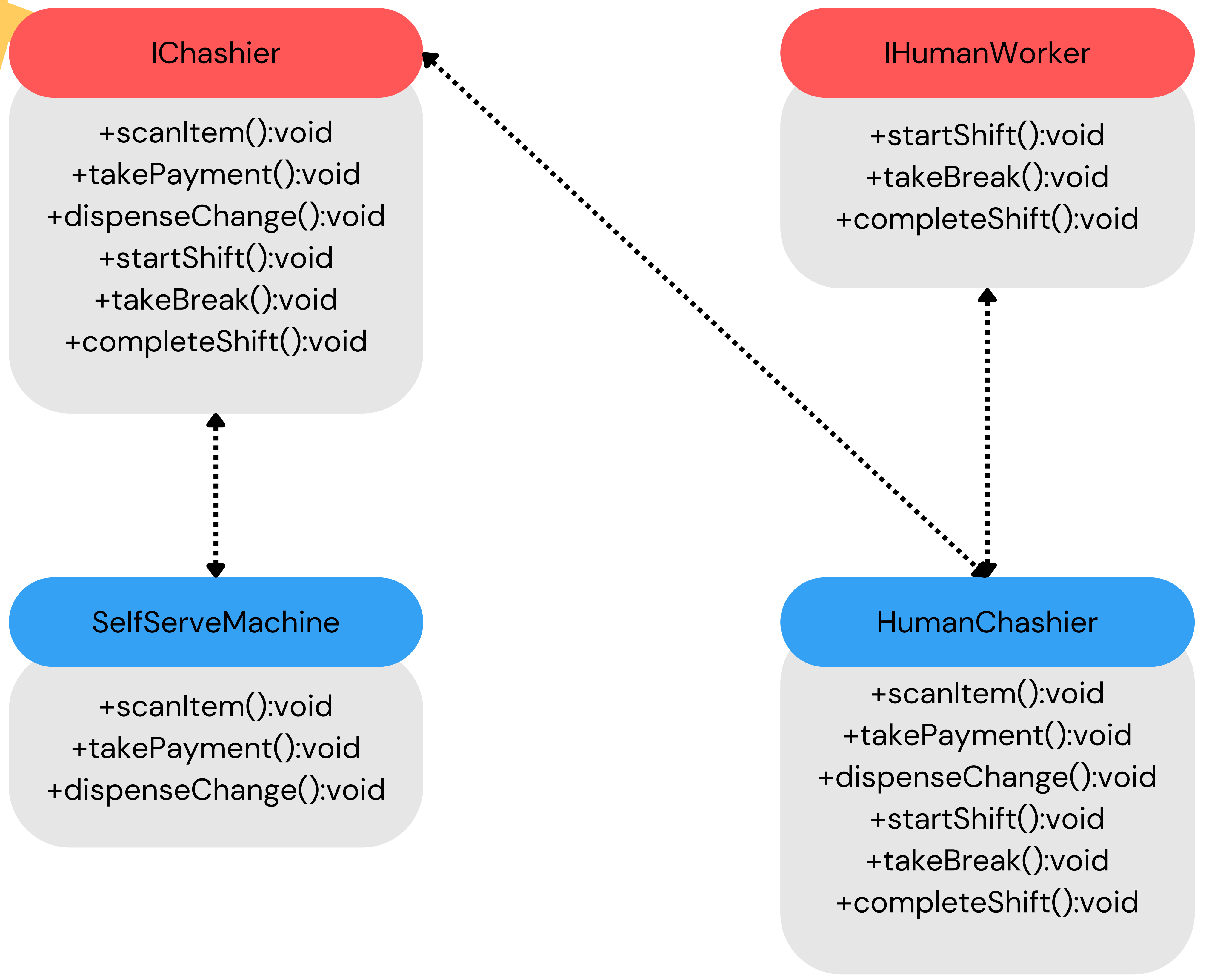
Interface Segregation Principle

Segregation means keeping things separated, and the Interface Segregation Principle is about separating the interfaces.
The principle states that many client-specific interfaces are better than one general-purpose interface. Clients should not be forced to implement a function they do no need.

LET'S START!




IChashier
+scanItem():void
+takePayment():void
+dispenseChange():void
+startShift():void
+takeBreak():void
+completeShift():void
SelfServeMachine
+scanItem():void
+takePayment():void
+dispenseChange():void
+startShift():void
+takeBreak():void
+completeShift():void
HumanChashier
+scanItem():void
+takePayment():void
+dispenseChange():void
+startShift():void
+takeBreak():void
+completeShift():void


IChashier
+scanItem():void
+takePayment():void
+dispenseChange():void
+startShift():void
+takeBreak():void
+completeShift():void
IHumanWorker
+startShift():void
+takeBreak():void
+completeShift():void
SelfServeMachine
+scanItem():void
+takePayment():void
+dispenseChange():void
HumanChashier
+scanItem():void
+takePayment():void
+dispenseChange():void
+startShift():void
+takeBreak():void
+completeShift():void

Back to Agenda Page

Dependency Inversion Principle

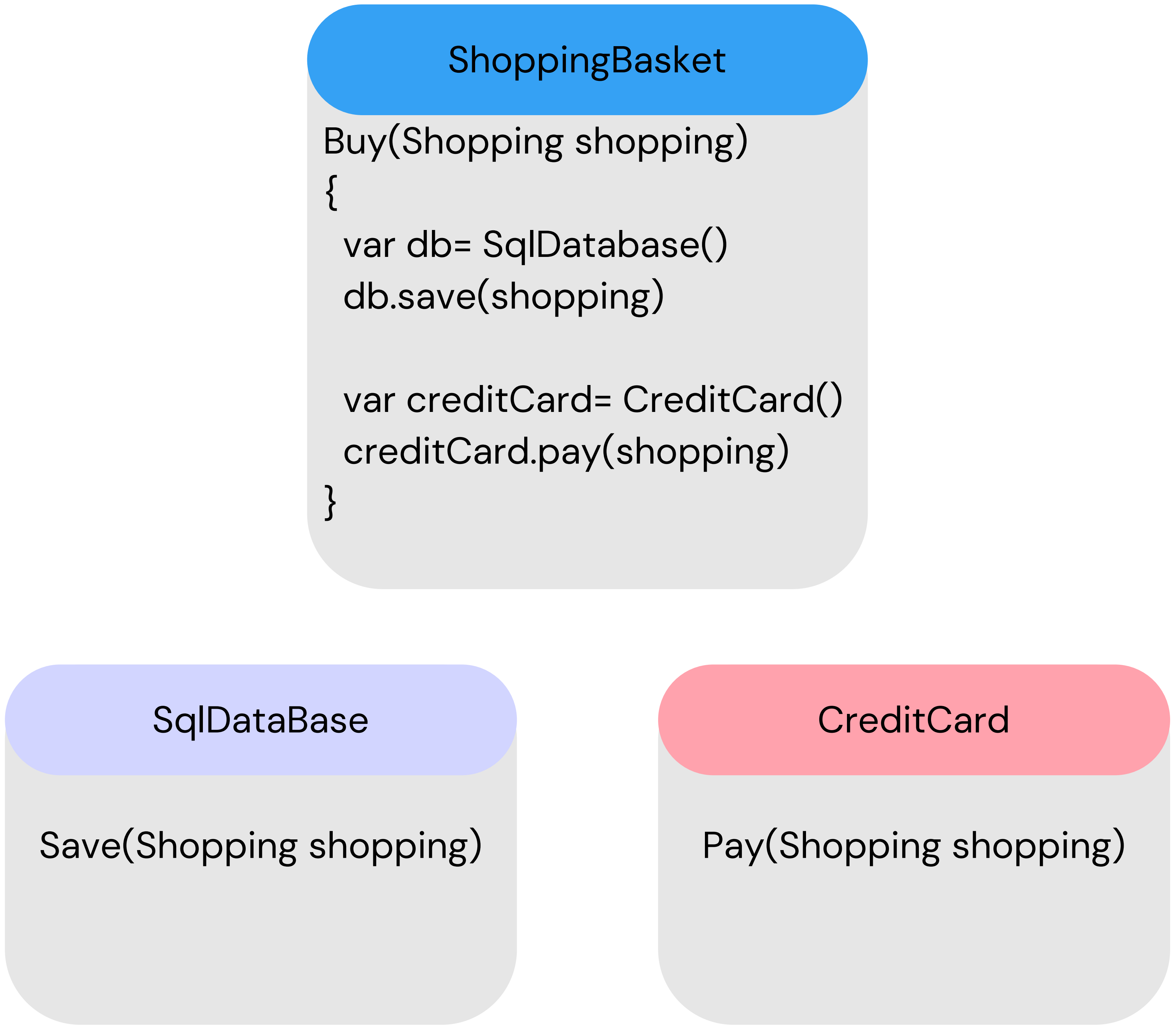
The Dependency Inversion principle states that our classes should depend upon interfaces or abstract classes instead of concrete classes and functions.

These two principles are indeed related and we have applied this pattern before while we were discussing the Open-Closed Principle.

LET'S START!

Back to Agenda Page

Shopping



ShoppingBasket

```
Buy(Shopping shopping)
{
  var db= SqlDatabase()
  db.save(shopping)

  var creditCard= CreditCard()
  creditCard.pay(shopping)
}
```

SqlDataBase

```
Save(Shopping shopping)
```

CreditCard

```
Pay(Shopping shopping)
```

Shopping

Back to Agenda Page

Shopping

```
Main()
 {
   var shopping1=ShoppingBasket(SqlDatabase(),CreditCard())
   var shopping2=ShoppingBasket(Server(),Paypal())
 }
```

ShoppingBasket

```
private readonly Persistence presistence
private readonly PaymentMethod paymentMethod

Buy(Shopping shopping)
 {
   presistence.save(shopping)
   paymentMethod.pay(shopping)
 }
```

SqlDataBase:Persistence

override Save(Shopping shopping)

Server:Persistence

override Save(Shopping shopping)

CreditCard:PaymentMethod

override Pay(Shopping shopping)

PayPal:PaymentMethod

override Pay(Shopping shopping)

Persistence

Save(Shopping shopping)

PaymentMethod

Pay(Shopping shopping)

Back to Agenda Page

Conclusion

SOLID principles are best practices that can help you write better code: cleaner, more maintainable and scalable.

As Robert C. Martin himself points out, "Getting a SOLID start" is not about rules, nor laws, nor absolute truths, but rather common sense solutions to common problems. They are heuristics, based on experience: "they have been observed to work in many cases; but there is no evidence that they always work, nor that they should always be followed."

Bibliography

The SOLID Principles of Object-Oriented Programming Explained in Plain English (freecodecamp.org)

SOLID: The First 5 Principles of Object Oriented Design | DigitalOcean