DEALING WITH OBJECTS IN OOP

A Narrative Journey from "Spaghetti Code" to Modern Structures

THE EVOLUTION OF SOFTWARE PROGRAMMING METHODS

1

1950s: SPAGHETTI CODE ERA

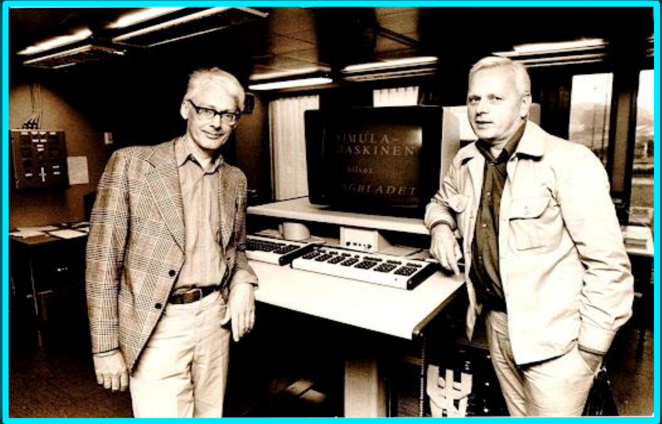
Before objects, programming was a linear chaos. Code jumped around unpredictably using GOTO statements. This "spaghetti code" was impossible to maintain. A small change in one place could break the entire system.

THE EVOLUTION OF SOFTWARE
PROGRAMMING METHODS

2

1960s:
THE SOLUTION:
SIMULA

Ole-Johan Dahl and Kristen Nygaard at the Norwegian Computing Center had a breakthrough. They invented Simula to simulate real-world systems (like exploding ships or queues). They introduced the concept of Classes and Objects to bundle data with behavior.



THE EVOLUTION OF SOFTWARE PROGRAMMING METHODS

3

1970s: Propagation and Purification

A team at Xerox PARC Laboratories, led by Alan Kay, developed the Smalltalk language. Smalltalk is considered the first purely object-oriented language. It established the concepts of encapsulation and polymorphism and introduced the term "object-oriented programming" itself.

THE EVOLUTION OF SOFTWARE
PROGRAMMING METHODS

4

1980s: Widespread Adoption

The C++ language emerged (by Bjarne Stroustrup). C++ combined the power and speed of the procedural C language with OOP concepts. This combination made OOP available for use in developing complex systems and led to its widespread industrial adoption (e.g., operating systems, desktop applications).

THE EVOLUTION OF SOFTWARE PROGRAMMING METHODS

5

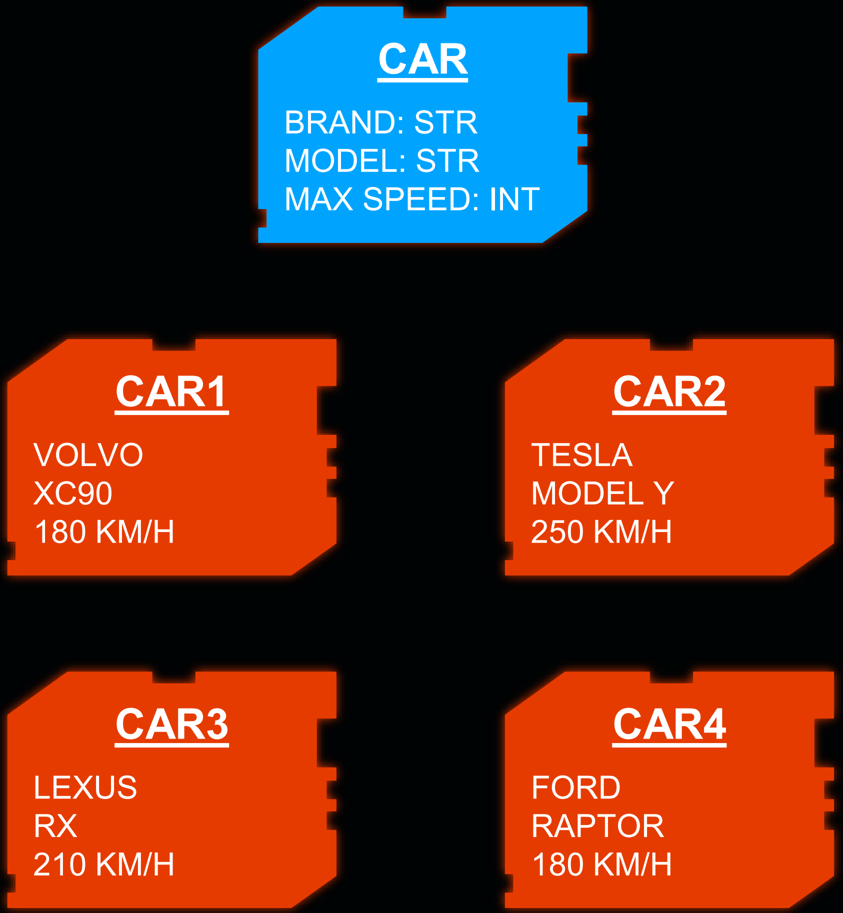
1990S & AFTER: OBJECTIFICATION MODEL DOMINANCE

Global Spread (Java, Python, C#): The emergence of languages like Java and Python, whose core design is entirely based on OOP. OOP has become the dominant and standard model in modern software development, from web applications to large enterprise systems to smartphone apps.

THE CORE CONCEPTS OF OOP

Think of a Class as a blueprint or a template. It defines the structure (Data/Properties) and capabilities (Methods/Functions) but doesn't exist as a tangible thing yet.

An Object is a specific instance created from the class blueprint. You can create thousands of objects from one class. Each has its own unique data (State) but shares the same structure.



CAR

BRAND: STR
MODEL: STR
MAX SPEED: INT

CAR1

VOLVO
XC90
180 KM/H

CAR2

TESLA
MODEL Y
250 KM/H

CAR3

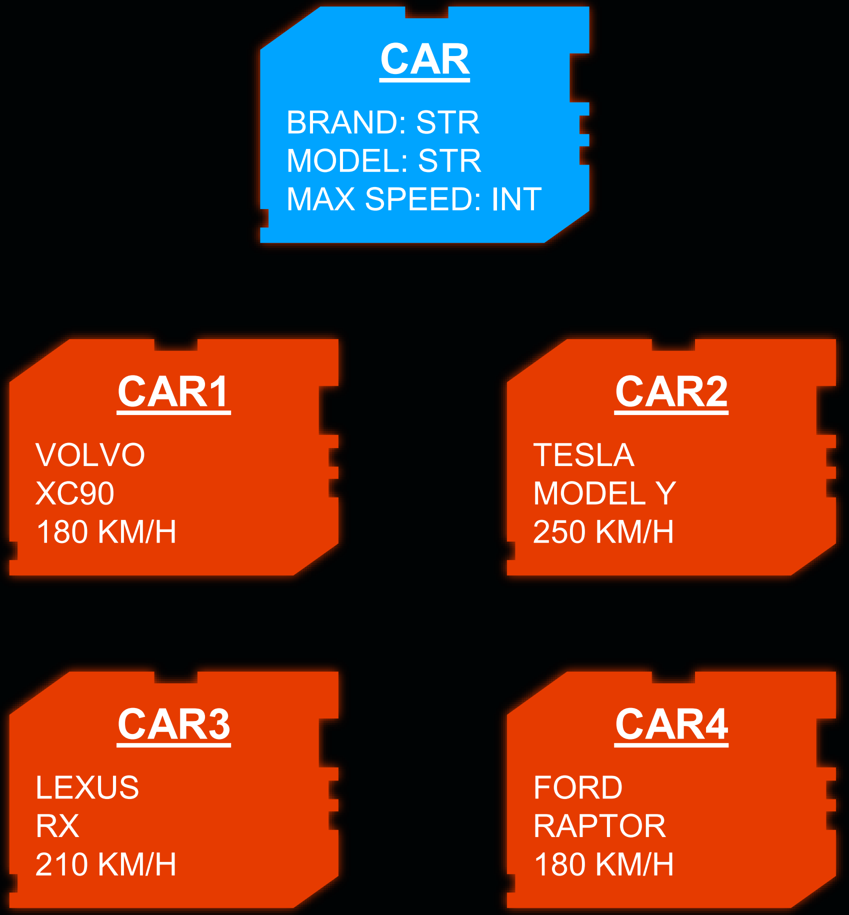
LEXUS
RX
210 KM/H

CAR4

FORD
RAPTOR
180 KM/H

PYTHON CODE:

```
class Car:
    def __init__(self, brand, model, max_speed):
        self.brand = brand       # STR
        self.model = model       # STR
        self.max_speed = max_speed  # INT
    def display_info(self):
        print(f"Brand: {self.brand}, Model:
    {self.model}, Max Speed: {self.max_speed}
    KM/H")

car1 = Car("VOLVO", "XC90", 180)
car2 = Car("TESLA", "MODEL Y", 250)
car3 = Car("LEXUS", "RX", 210)
car4 = Car("FORD", "RAPTOR", 180)

print("CAR1 Details:")
car1.display_info()
print("\nCAR2 Details:")
car2.display_info()
print("\nCAR3 Details:")
car3.display_info()
print("\nCAR4 Details:")
car4.display_info()
```



CAR

BRAND: STR
MODEL: STR
MAX SPEED: INT

CAR1

VOLVO
XC90
180 KM/H

CAR2

TESLA
MODEL Y
250 KM/H

CAR3

LEXUS
RX
210 KM/H

CAR4

FORD
RAPTOR
180 KM/H

WHAT IF WE HAVE 1000 CARS?

CAR1

VOLVO
XC90
180 KM/H

CAR2

TESLA
MODEL Y
250 KM/H

CAR3

LEXUS
RX
210 KM/H

CAR4

FORD
RAPTOR
180 KM/H

CAR5

MAZDA
3
190 KM/H

CAR6

AUDI
A4
240 KM/H

CAR7

SUBARU
OUTBACK
200 KM/H

CAR8

HYUNDAI
ELANTRA
185 KM/H

CAR9

PORSCHE
911
320 KM/H

CAR….

…
…
…

WHAT IF WE HAVE
1000 CARS?

CAR1 VOLVO XC90 180 KM/H	**CAR2** TESLA MODEL Y 250 KM/H	**CAR3** LEXUS RX 210 KM/H	**CAR4** FORD RAPTOR 180 KM/H	**CAR5** MAZDA 3 190 KM/H
CAR6 AUDI A4 240 KM/H	**CAR7** SUBARU OUTBACK 200 KM/H	**CAR8** HYUNDAI ELANTRA 185 KM/H	**CAR9** PORSCHE 911 320 KM/H	**CAR….** … … …

Creating objects like car1, car2... car1000 is inefficient. So, we use Object Arrays, Object Arrays allow us to store multiple objects in a single structured list, indexed by number. We can loop through the list to process all cars info at once

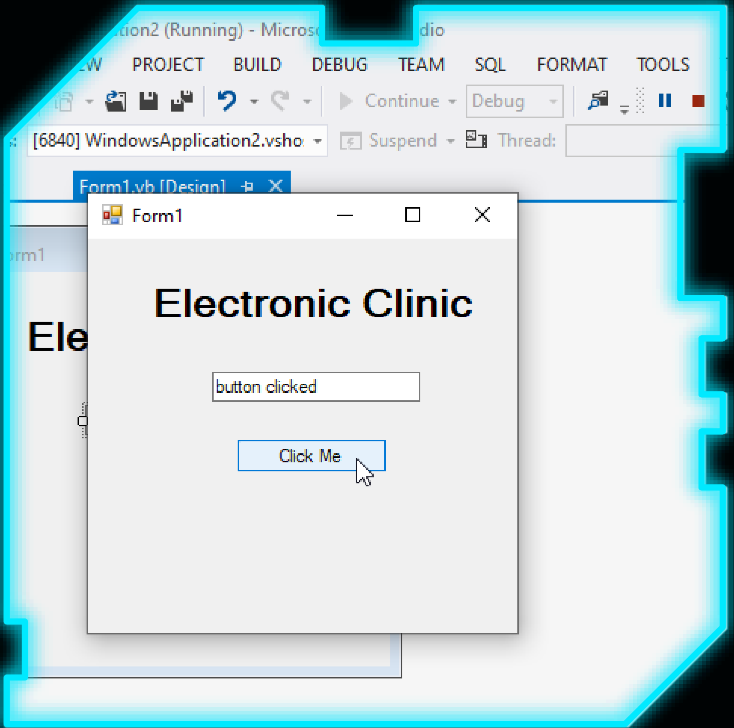
PYTHON CODE:

```python
class Car:
    def __init__(self, brand, model, max_speed):
        self.brand = brand
        self.model = model
        self.max_speed = max_speed

def main():
    cars_array = [
        Car("VOLVO", "XC90", 180),  # CAR 1
        Car("TESLA", "MODEL Y", 250), # CAR 2
        Car("LEXUS", "RX", 210),  # CAR 3
        Car("FORD", "RAPTOR", 180),  # CAR 4
        Car("MAZDA", "3", 190),  # CAR 5
        Car("AUDI", "A4", 240),  # CAR 6
        Car("SUBARU", "OUTBACK", 200),  # CAR 7
        Car("HYUNDAI", "ELANTRA", 185),  # CAR 8
        Car("PORSCHE", "911", 320)  # CAR 9
    ]

    for car in cars_array:
        print(f"Brand: {car.brand}, Model: {car.model}, Speed: {car.max_speed} KM/H")

if __name__ == "__main__":
    main()
```

EVERYTHING IS AN OBJECT

In modern Windows apps, every button, text box, and window is an object.

You can create arrays of Buttons just like arrays of Cars to modify them programmatically(e.g., changing colors or text in a loop).


tion2 (Running) - Micros dio
W PROJECT BUILD DEBUG TEAM SQL FORMAT TOOLS
Continue
Debug
[6840] WindowsApplication2.vsho
Suspend
Thread:
Form1.vb [Design]
rm1
Ele
Form1
Electronic Clinic

button clicked

FROM CHAOS TO STRUCTURE

We journeyed from the "spaghetti code" of the 1950s to the organized, reusable Object-Oriented paradigms of today.

Whether in Python or VB.NET or any OOP language, Objects allow us to build complex, maintainable, and scalable software.

THANKS!

PREPARED BY:

AHMED SHEHATA SAID
AHMED TAMER ELSAYED
AYA MAGDY ABDULLAH
ESRAA SALAH MOHAMED
SARAH ESSAM ELDIN

UNDER SUPERVISION OF: DR. AMANI AL-JAMAL