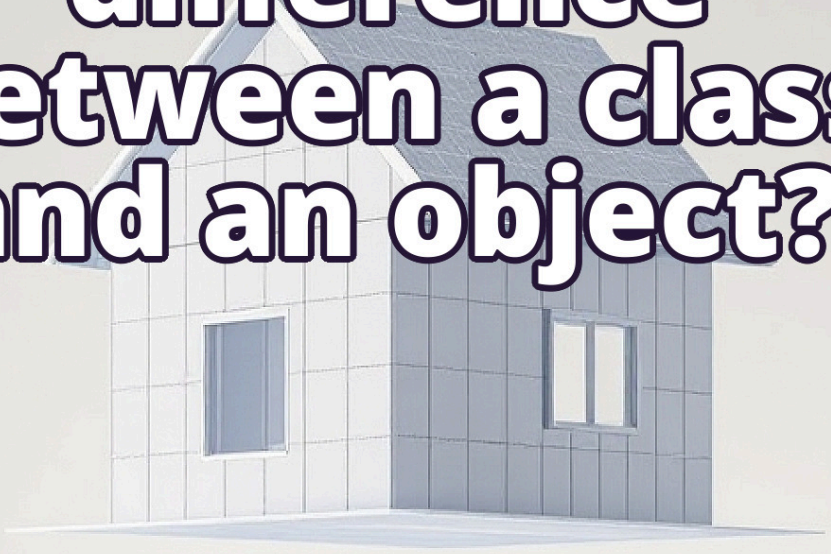
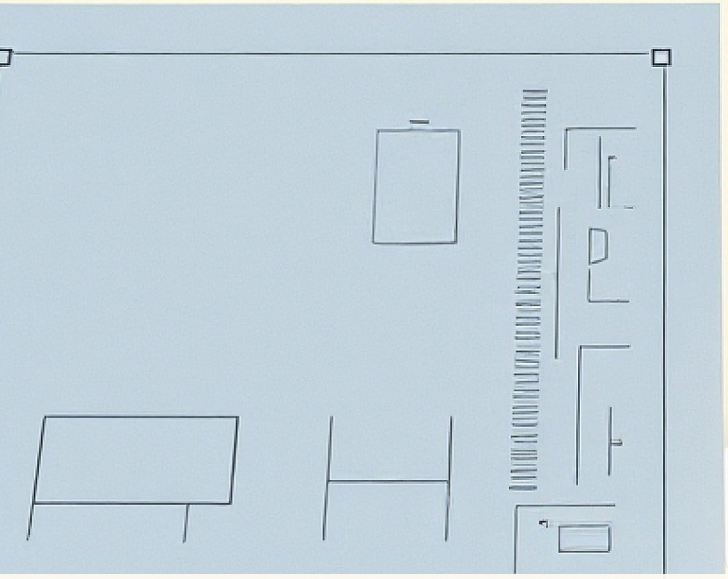
What is the difference between a class and an object?

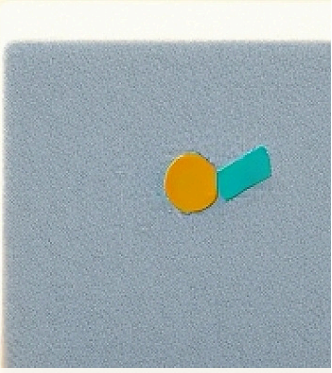




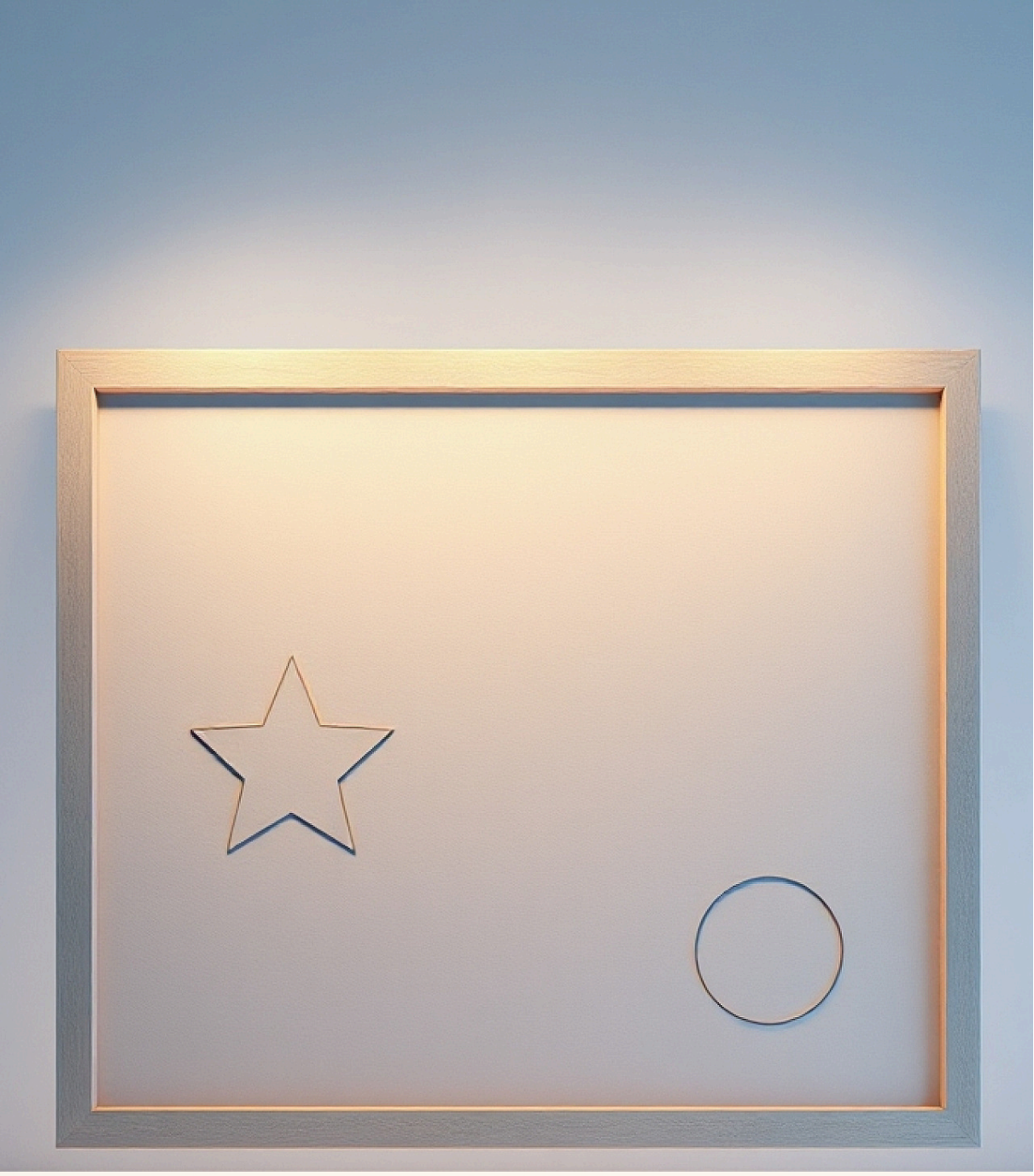
www.yzokul.com.tr

What is the difference between a class and an object?





The question is simple: what is the difference between a class and an object? The short answer is that a class is a definition, and an object is a specific instance made from it. Think of a class as a plan and an object as a real thing built from that plan. This book explains each idea step by step so you can see the difference clearly.



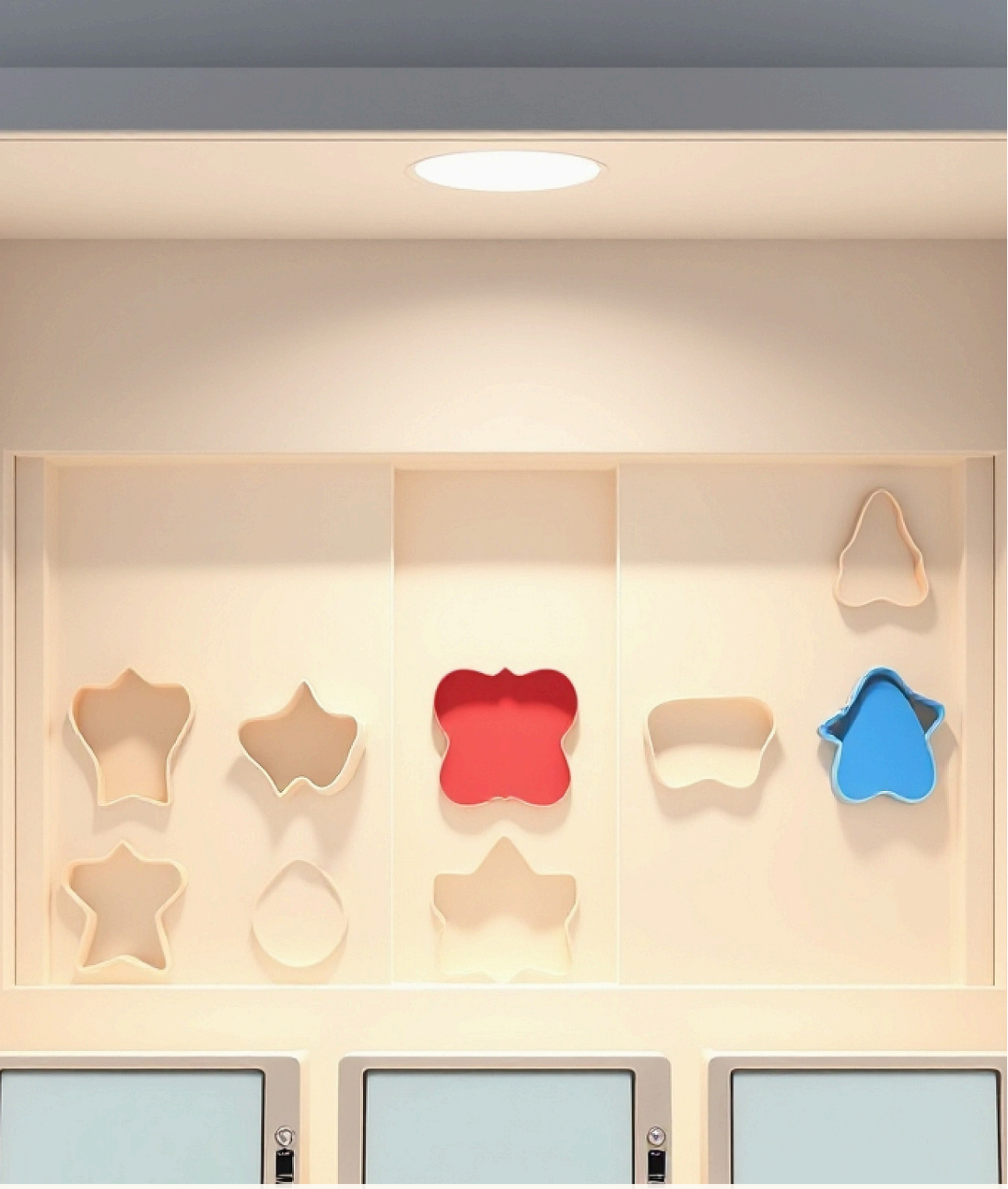
A class is a structure that describes which properties and behaviors something of a certain type will have. It is a description, not a real thing you can use directly. For example, a class called Player might describe that a player has a name and a score. The class itself does not contain any actual player data; it only says what data and actions players will have.



An object is a specific instance created from a class. It is a real thing that exists and has its own values for the properties the class describes. For example, an object made from the Player class might have the name Ana and a score of 10. The object is the actual player you can use in your program.



The core relationship is definition versus instance. A class is a definition, and an object is an instance of that definition. One describes what something will be like, and the other is a real thing that exists. For example, the Player class defines what a player is, and an object like Ana with 10 points is a specific player.



A class lists properties, which are the pieces of information objects of that type will have. For example, a Player class lists a name and a score. These properties are like slots that each object will fill with its own values. The class only says which properties exist, not what values they hold.



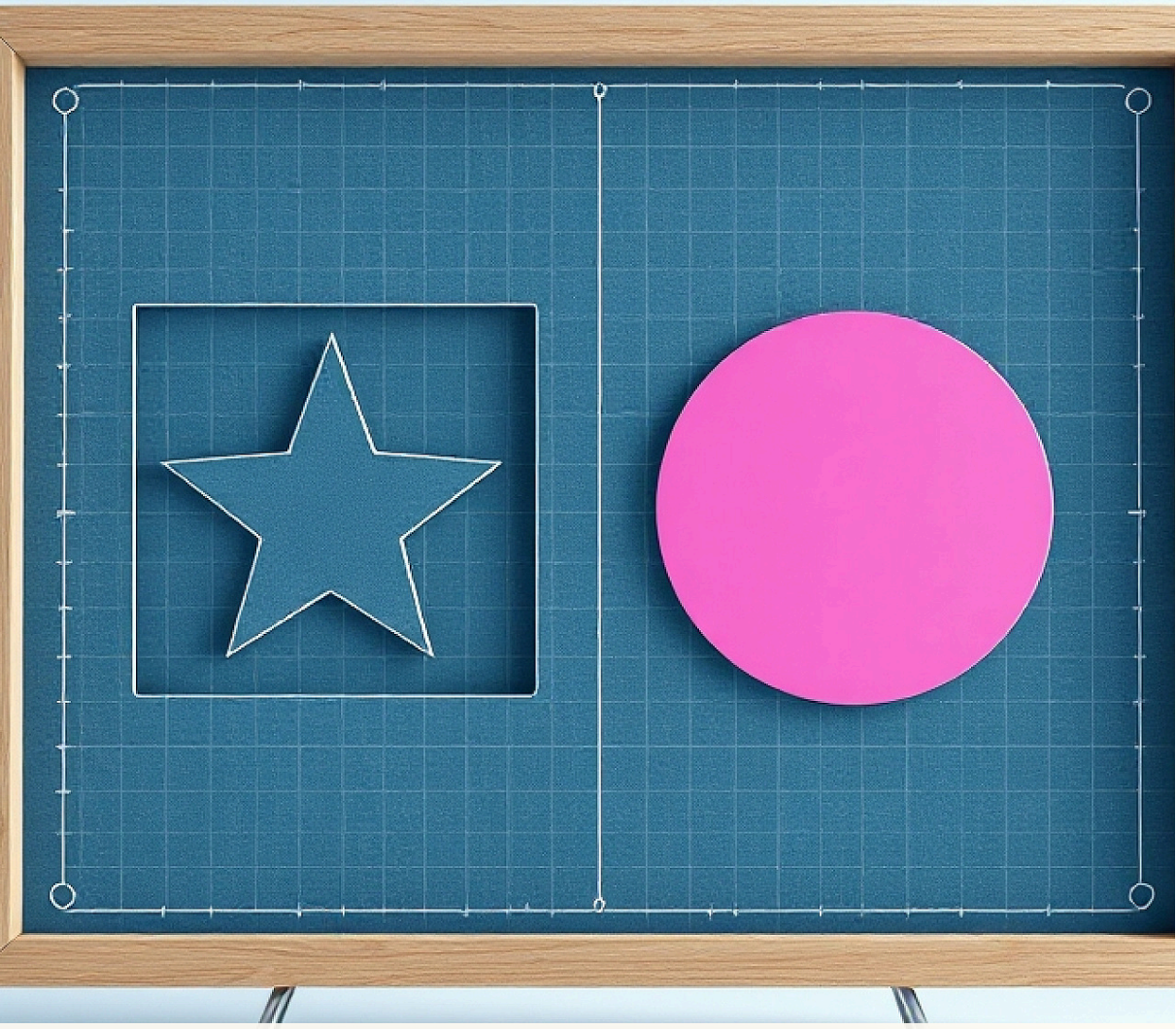
A class also describes behaviors, which are the actions objects of that type can do. For example, a Player can gain points or show its score. These behaviors are written once in the class and are available to all objects made from it. The class defines what actions are possible, but each object performs them with its own data.



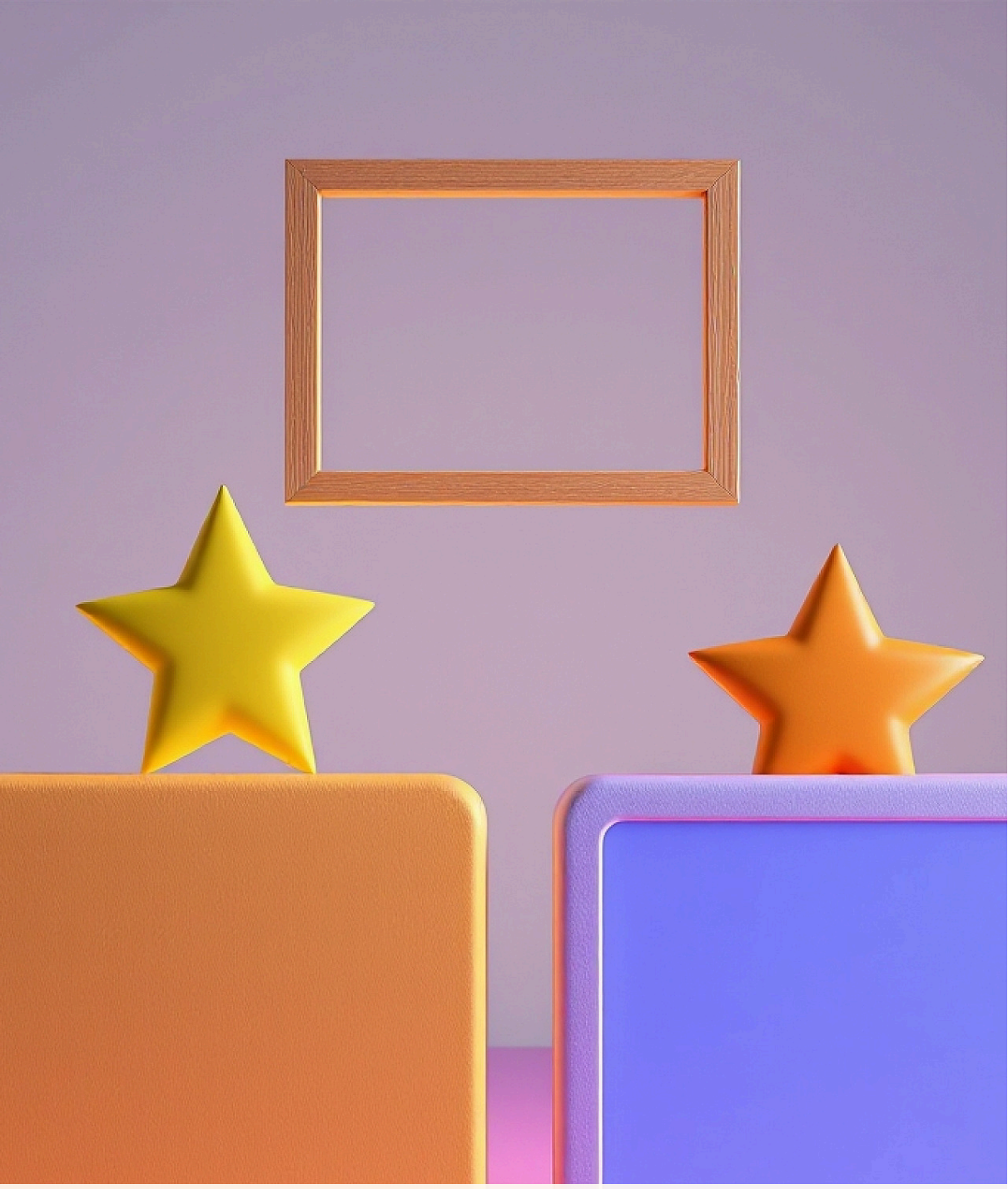
The recipe analogy makes this clear. A recipe is like a class: it describes how to make a dish, including ingredients and steps. Each dish made from the recipe is like an object: it is a real, specific thing you can eat. The recipe itself is not food; it is only instructions.



One recipe can produce many different dishes, just as one class can produce many objects. Each dish is separate even though it comes from the same recipe. For example, you can make several cakes from one recipe, and each cake is its own object with its own size and decorations. The recipe stays the same.



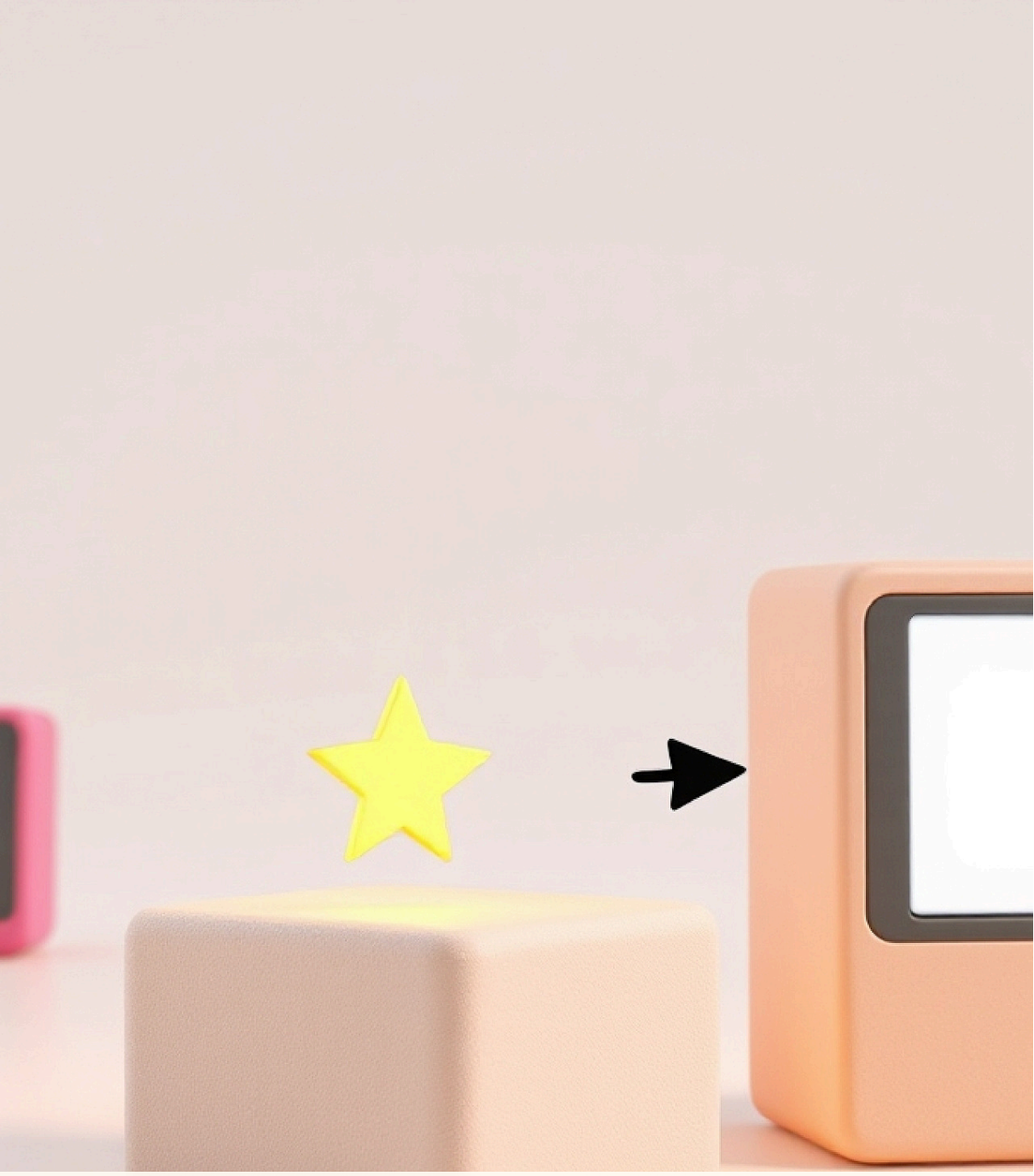
Consider a Player class that describes a name and a score. This class is only the description, not any actual player. It tells you that every player will have a name and a score, but it does not give any specific name or score. The class is the blueprint, not the building.



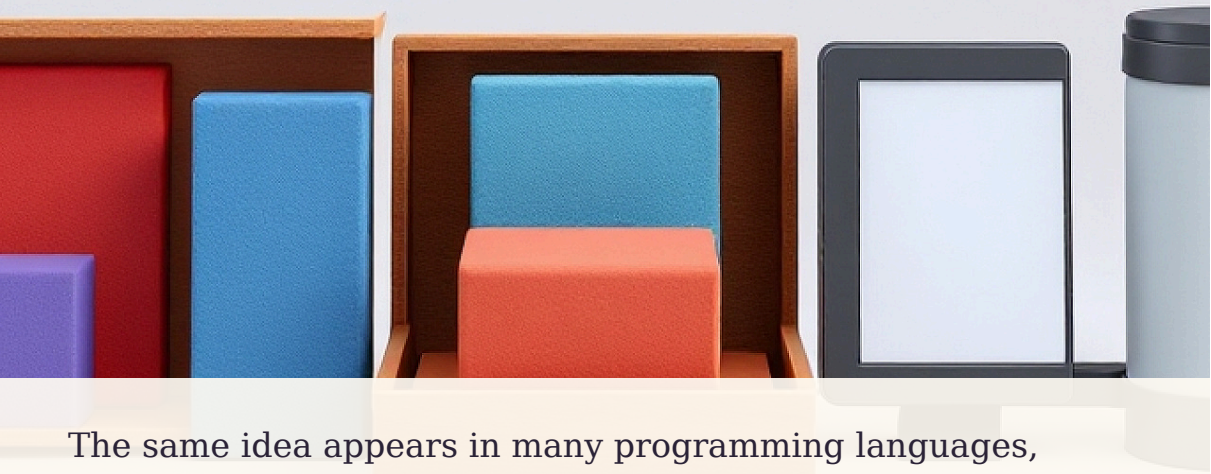
Creating player objects means making specific instances from the Player class. For example, one player object is named Ana with 10 points, and another is named Ben with 25 points. Each object is created separately and has its own name and score. The class is used as a template to produce these objects.



Objects from the same class can hold different values. The class is shared, but the data inside each object belongs to that object alone. For example, Ana has 10 points and Ben has 25 points; their scores are independent. Changing Ana's score does not affect Ben's score because each object has its own copy of the properties.



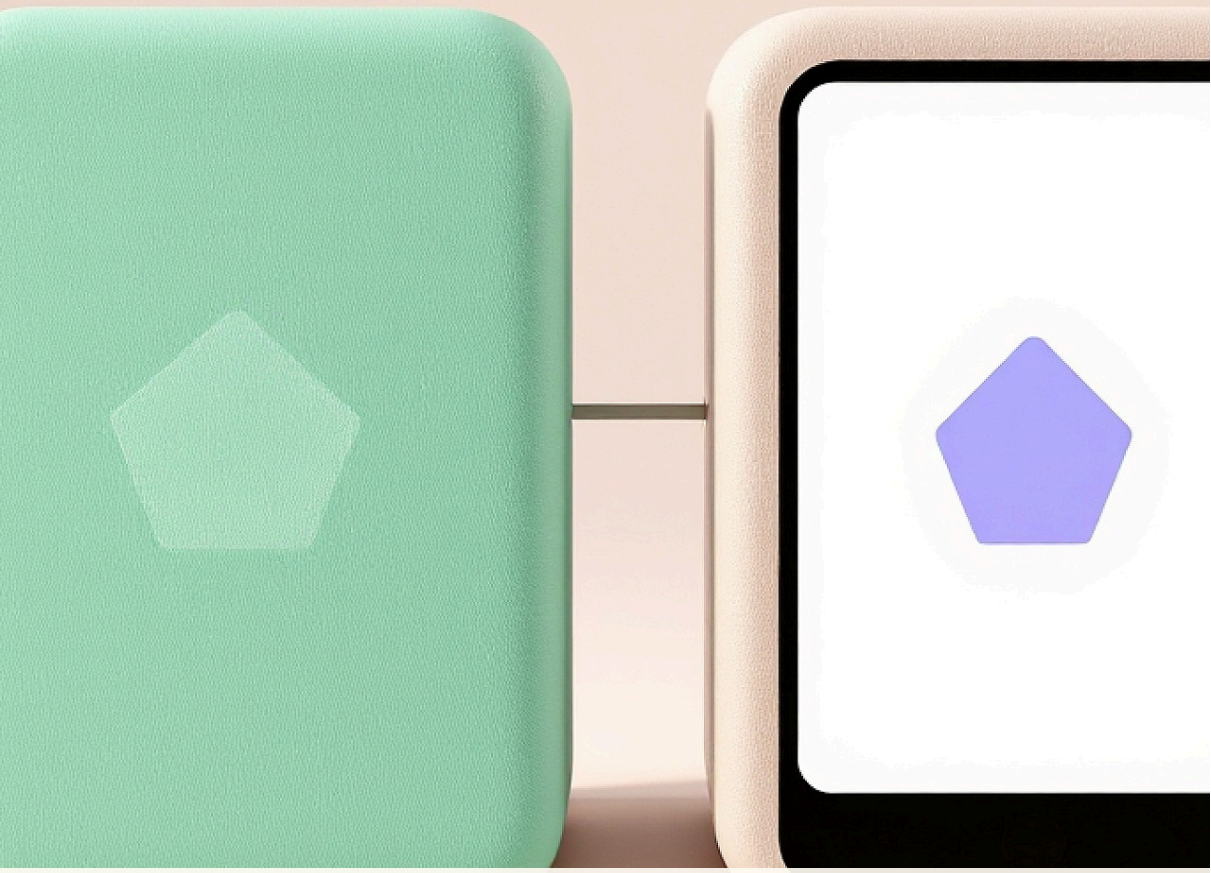
Changing one object's values does not change the other objects. Each object is separate and keeps its own information. For example, if you add 5 points to Ana's score, Ben's score stays the same. This separation is a key feature of objects: they are independent instances even when they come from the same class.



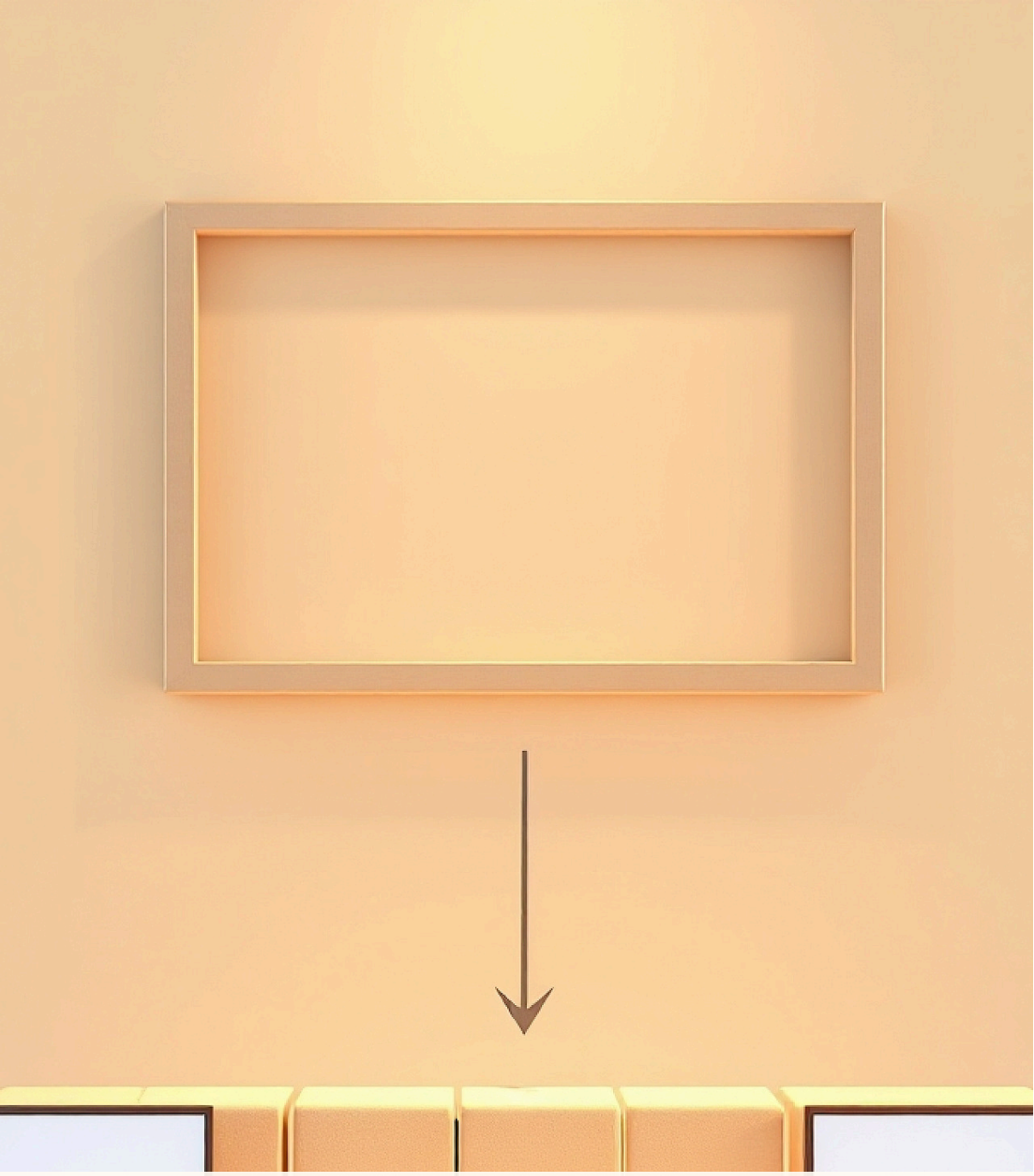
The same idea appears in many programming languages, though the details vary. In every case, the basic relationship is between a definition and an instance of that definition. Some languages use different words, like struct or record, but the concept is the same: a class describes, and an object is a real thing made from it.



A common mistake is treating the class itself as if it were a usable object. The class only describes; you must create an object to have a real thing. For example, you cannot give the Player class a score directly; you must create a player object and then set its score. The class is not a player.



Another common mistake is thinking all objects from one class are identical copies. They share the same description but each holds its own separate values. For example, two player objects can have different names and scores, and changing one does not change the other. Objects are independent instances, not clones.



To sum up, a class is a definition of properties and behaviors, and an object is a specific instance created from it. The key thing to remember is definition versus instance, like a recipe and the dishes made from it. Keep this relationship in mind, and you will always know the difference between a class and an object.