



PUC Minas



Adapter

Produção de Jogos 4 - Inteligência Artificial

Produção de Jogos 4 - Padrões		
Design Patterns (Padrões de Projeto)		
Chapão	Estrutura	Comportamental
Singleton	Flyweight	Strategy
Factory Method	Composite	State
Builder	Decorator	Observer
Abstract Factory	Adapter	Command
Prototype	Facade	Iterator
	Bridge	Mediator
	Proxy	Memento
		Interpreter
		Template Methode
		Visitor
		Chain of responsibility

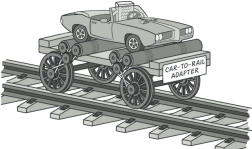
Produção de Jogos 4 - Adapter

Adapter / Adaptador

O padrão **Adapter** converte a interface de uma classe para outra interface que o cliente espera encontrar.
O adaptador permite que classes com interfaces incompatíveis trabalhem juntas.

Uso:

Há dois tipos de Adapter: Adaptador de objeto e adaptador de classe.
O adaptador de classe herda das duas classe que se quer adaptar (nem sempre possível pois alguma linguagens não permitem herança de múltiplas classes, ex.: java).



<https://refactoring.guru/design-patterns/adapter>

Produção de Jogos 4 - Adapter

Adapter / Adaptador

Prática:
Ajusta um objeto para funcionar no formato exigido por outros.
Um Adapter envelopa um objeto para mudar sua interface;

Carro

+Move()

Tem um (composição)

CarroAdapter: Trem


+CarroAdapter(Carro)

+Move()

Trem

+Move()

É um (Realização)



Produção de Jogos 4 - Adapter

```
using System;

public interface ICarro {
    void MoverEmRua();
}

public interface ITrem {
    void MoverEmTrilho();
}

public class Ford: ICarro{
    public void MoverEmRua(){
        Console.WriteLine("Mover carro");
    }
}

public class Metro: ITrem {
    public virtual void MoverEmTrilho(){
        Console.WriteLine("Mover trem");
    }
}

public class CarroAdapter: Metro {
    ICarro carro;

    public CarroAdapter(ICarro carro){
        this.carro = carro;
    }

    public override void MoverEmTrilho(){
        carro.MoverEmRua();
    }
}

public class Program {
    public static void Main() {
        Ford k = new Ford();
        k.MoverEmRua();
        Metro m = new CarroAdapter(k);
        m.MoverEmTrilho();
    }
}
```

Dados JSON

Estrutura de Dados JSON

- O formato JSON (Jay-son)(acrônimo para "JavaScript Object Notation") foi originalmente criado por Douglas Crockford e é descrito no RFC 4627, com media-type .json.
- É um formato leve para intercâmbio de dados computacionais.
- JSON é um subconjunto da notação de objeto de JavaScript, mas seu uso não requer JavaScript exclusivamente.
- A simplicidade de JSON (fácil de escrever, interpretar) tem resultado em seu uso difundido, especialmente como uma alternativa para XML em AJAX.

Dados JSON

Sintaxe:
Um objeto começa e termina com chaves:
{ }

Cada item possui uma chave e valor (como em um dicionário)

"nome": "João"	- String
"idade": 40	- Number
"altura": 1.7	- Number
"casado": true	- Boolean
"vazio": null	- Vazio
Object - pode ser	- JSON
"notas": [10, 8.5, 9]	- Array

Exemplo

```
{
  "position": { "x":5.0, "y":0.0, "z":0.0 },
  "health": 80
}
```

<https://jsonformatter.curiousconcept.com/>

Dados JSON

Sintaxe:
Caracteres Especiais

1.	"	Double quotation.
2.	\	Backslash.
3.	/	Forward slash
4.	B	Backspace.
5.	f	Form feed.
6.	n	New line.
7.	r	Carriage return.
8.	t	Horizontal tab.
9.	u	Four hexadecimal digits.

<https://jsonformatter.curiousconcept.com/>

Dados JSON

```
{ "Alunos": [
  { "nome": "João",
    "idade": 12,
    "notas": [ 8, 9, 7 ]
  },
  { "nome": "Maria",
    "idade": 13,
    "notas": [ 8, 10, 7 ]
  },
  { "nome": "Pedro",
    "idade": 11,
    "notas": [ 10, 10, 9 ]
  }
]
```

Dados JSON

Exemplo:

```
Assets > SaveGame > PlayerData.cs > ...
1 using UnityEngine;
2 using System;
3
4 [Serializable]
5 public class PlayerData {
6     public Vector3 position;
7     public Vector3 rotation;
8 }
```

```
Assets > SaveGame > Player.cs > ...
1 using UnityEngine;
2
3 public class Player : MonoBehaviour {
4
5     void Update() {
6         if (Input.GetKeyDown(KeyCode.S)) Teste();
7     }
8
9     void Teste(){
10         PlayerData data = new PlayerData();
11         data.position = transform.position;
12         data.rotation = transform.rotation.eulerAngles;
13         string s = JsonUtility.ToJson(data);
14         Debug.Log(s);
15     }
16 }
```

Produção de Jogos 4 - Adapter

Adapter / Adaptador (Save e Load Game)

```
Assets > SaveGame > PlayerData.cs > ...
1 using UnityEngine;
2 using System;
3
4 [Serializable]
5 public class PlayerData {
6     public Vector3 position;
7     public Vector3 rotation;
8 }
```

```
Assets > SaveGame > Player.cs > ...
1 using UnityEngine;
2 using System;
3
4 public class Player : MonoBehaviour {
5
6     void Update() {
7         if (Input.GetKeyDown(KeyCode.S)) SaveManager.Save(this);
8         if (Input.GetKeyDown(KeyCode.L)) SaveManager.Load(this);
9     }
10 }
```

Produção de Jogos 4 - Adapter

Adapter / Adaptador (Save e Load Game)

```
Assets > SaveGame > PlayerData.cs > ...
1 using UnityEngine;
2 using System;
3
4 [Serializable]
5 public class PlayerData {
6     public Vector3 position;
7     public Vector3 rotation;
8 }
```

```
Assets > SaveGame > Player.cs > ...
1 using UnityEngine;
2 using System;
3
4 public class Player : MonoBehaviour {
5
6     void Update() {
7         float h = Input.GetAxis("Horizontal");
8         float v = Input.GetAxis("Vertical");
9         transform.Translate(h * 6f * Time.deltaTime, v * 6 * Time.deltaTime, 0);
10         if (Input.GetKeyDown(KeyCode.S)) SaveManager.Save(this);
11         if (Input.GetKeyDown(KeyCode.L)) SaveManager.Load(this);
12     }
13 }
```

Produção de Jogos 4 - Adapter

Adapter / Adaptador (Save e Load Game)

PlayerDataAdapter.cs

Assets > SaveGame > PlayerDataAdapter.cs > PlayerDataAdapter

```
1 using UnityEngine;
2
3
4 public class PlayerDataAdapter : PlayerData
5 {
6     public PlayerDataAdapter(Player player){
7         position = player.transform.position;
8         rotation = player.transform.rotation.eulerAngles;
9     }
10
11     public static void DataToPlayer(Player player, PlayerData data){
12         player.transform.position = data.position;
13         player.transform.rotation = Quaternion.Euler(data.rotation);
14     }
15 }
```

Produção de Jogos 4 - Adapter

Adapter / Adaptador (Save e Load Game)

SaveManager.cs

Assets > SaveGame > SaveManager.cs > ...

```
1 using UnityEngine;
2 using System.IO;
3
4 public class SaveManager {
5     public static void Save(Player player){
6         PlayerData playerData = new PlayerDataAdapter(player);
7         string s = JsonUtility.ToJson(playerData);
8         File.WriteAllText(Application.dataPath + "/save.json", s);
9     }
10
11     public static void Load(Player player){
12         string s = File.ReadAllText(Application.dataPath + "/save.json");
13         PlayerData playerData = JsonUtility.FromJson<PlayerData>(s);
14         PlayerDataAdapter.DataToPlayer(player, playerData);
15     }
16 }
```

Produção de Jogos 4 - Adapter

Adapter / Adaptador (Save e Load Game)

```
using UnityEngine;
using System;

[Serializable]
public class PlayerData {
    public Vector3 position;
    public Vector3 rotation;
}
```