
FLYWEIGHT
ALUNOS: EDILSON JÚNIOR, ISHTEFANI VITÓRIA, JÚLIO EMERICK
PLAY
MENU
EXIT

PROBLEMA

EM UM JOGO, SE CADA **INIMIGO** FOR UMA INSTÂNCIA SEPARADA, CONSUMIRÁ UMA QUANTIDADE SIGNIFICATIVA DE MEMÓRIA.

COMO MELHORAR ISSO?

EXIT

O FLYWEIGHT SOLUCIONA ISSO

COM O FLYWEIGHT É POSSÍVEL COMPARTILHAR PARTE DOS DADOS ENTRE VÁRIOS OBJETOS SEMELHANTES, EM VEZ DE DUPLICÁ-LOS EM CADA OBJETO.

COMO ISSO É FEITO?

1. IDENTIFIQUE AS PARTES DO OBJETO QUE PODEM SER COMPARTILHADAS
2. CRIE UMA INTERFACE QUE DEFINE OS MÉTODOS NECESÁRIOS PARA OS OBJETOS COMPARTILHADOS
3. CRIE CLASSES QUE IMPLEMENTAM A INTERFACE

EXIT

COMO ISSO É FEITO?

4 CRIE UMA FÁBRICA PARA GERENCIAR A CRIAÇÃO E RECUPERAÇÃO DOS OBJETOS

5 SOLICITE OS OBJETOS À FÁBRICA QUANDO PRECISAR DELES

EXIT

GERENCIADOR DE DOCUMENTOS

```java
// Classe Document sem o Flyweight
class Document {
    private String author;
    private String content;

    public Document(String author, String content) {
        this.author = author;
        this.content = content;
    }

    public void printContent() {
        System.out.println("Documento: " + content + " | Autor: " + author);
    }
}

// Client
public class Main {
    public static void main(String[] args) {
        Document doc1 = new Document("Autor 1", "Conteúdo 1");
        doc1.printContent();

        Document doc2 = new Document("Autor 2", "Contento 2");
        doc2.printContent();

        Document doc3 = new Document("Autor 1","Contento 3");
        doc3.printContent();

        System.out.println();

        System.out.println("O Documento 2 foi criado com base em um objeto já existente? " + (doc2 == doc1));
        System.out.println("O Documento 3 foi criado com base em um objeto já existente? " + (doc3 == doc1));
    }
}
```

EXIT

APLICAÇÃO

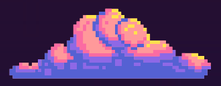
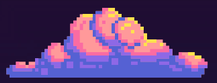
```java
import java.util.HashMap;
import java.util.Map;

// 1. Flyweight (Interface)
interface Document {
    void printContent(String author);
}

```

```java

// 2. ConcreteFlyweight
class ConcreteDocument implements Document {
    private String content;

    public ConcreteDocument(String content) {
        this.content = content;
    }

    public void printContent(String author) {
        System.out.println("Documento: " + content + " | Autor: " + author);
    }
}

```

EXIT

APLICAÇÃO

```
21
22  // 3. FlyweightFactory
23  class DocumentFactory {
24      private Map<String, Document> documents = new HashMap<>();
25
26      public Document getDocument(String content) {
27          // Verifica se o objeto já existe na memória
28          if (documents.containsKey(content)) {
29              return documents.get(content);
30          } else {
31              // Caso contrário, cria um novo objeto e o armazena na memória
32              Document document = new ConcreteDocument(content);
33              documents.put(content, document);
34              return document;
35          }
36      }
37
38      public int getCacheSize() {
39          return documents.size();
40      }
41  }
42
```

APLICAÇÃO

```java
// 4. Cliente
public class Main {
    public static void main(String[] args) {
        DocumentFactory documentFactory = new DocumentFactory();

        Document doc1 = documentFactory.getDocument("Conteúdo 1");
        doc1.printContent("Autor 1");

        Document doc2 = documentFactory.getDocument("Conteúdo 2");
        doc2.printContent("Autor 2");

        Document doc3 = documentFactory.getDocument("Conteúdo 1");
        doc3.printContent("Autor 3");

        System.out.println();

        int cacheSize = documentFactory.getCacheSize();

        System.out.println("Total de objetos criados: " + cacheSize);
        System.out.println("O Documento 2 foi criado com base em um objeto já existente? " + (doc2 == doc1));
        System.out.println("O Documento 3 foi criado com base em um objeto já existente? " + (doc3 == doc1));
    }
}
```

EXIT

ANTES // DEPOIS

```java
// Classe Document sem o Flyweight
class Document {
    private String author;
    private String content;

    public Document(String author, String content) {
        this.author = author;
        this.content = content;
    }

    public void printContent() {
        System.out.println("Documento: " + content + " | Autor: " + author);
    }
}

// Client
public class Main {
    public static void main(String[] args) {
        Document doc1 = new Document("Autor 1", "Conteúdo 1");
        doc1.printContent();

        Document doc2 = new Document("Autor 2", "Contento 2");
        doc2.printContent();

        Document doc3 = new Document("Autor 1","Contento 3");
        doc3.printContent();

        System.out.println();

        System.out.println("O Documento 2 foi criado com base em um objeto já existente? " + (doc
        System.out.println("O Documento 3 foi criado com base em um objeto já existente? " + (doc
    }
}
```

```java
import java.util.HashMap;
import java.util.Map;

// 1. Flyweight (Interface)
interface Document {
    void printContent(String author);
}

// 2. ConcreteFlyweight
class ConcreteDocument implements Document {
    private String content;

    public ConcreteDocument(String content) {
        this.content = content;
    }

    public void printContent(String author) {
        System.out.println("Conteúdo: " + content + " | Autor: " + author);
    }
}

// 3. FlyweightFactory
class DocumentFactory {
    private Map<String, Document> documents = new HashMap<>();

    public Document getDocument(String content) {
        // Verifica se o objeto já existe na memória
        if (documents.containsKey(content)) {
            return documents.get(content);
        } else {
            // Caso contrário, cria um novo objeto e o armazena na memória
            Document document = new ConcreteDocument(content);
            documents.put(content, document);
            return document;
        }
    }

    public int getCacheSize() {
        return documents.size();
    }
}

// 4. Cliente
public class Main {
    public static void main(String[] args) {
        DocumentFactory documentFactory = new DocumentFactory();

        Document doc1 = documentFactory.getDocument("Documento 1");
        doc1.printContent("Autor 1");

        Document doc2 = documentFactory.getDocument("Documento 2");
        doc2.printContent("Autor 2");

        Document doc3 = documentFactory.getDocument("Documento 1");
        doc3.printContent("Autor 3");

        int cacheSize = documentFactory.getCacheSize();

        System.out.println("Total de objetos criados: " + cacheSize);
        System.out.println("O Documento 2 foi criado com base em um objeto já existente? " + (doc2 == doc1));
        System.out.println("O Documento 3 foi criado com base em um objeto já existente? " + (doc3 == doc1));
    }
}
```

EXIT

PONTOS POSITIVOS

ECONOMIA DE MEMRÓRIA

O FLYWEIGHT PERMITE REDUZIR O USO DA MEMÓRIA, COMPARTILHANDO PARTES COMUNS DO ESTADO ENTRE VÁRIAS INSTANCIAS.

FLEXIBILIDADE

PERMITE QUE OS OBJETOS SEJAM COMPARTILHADOS ENTRE DIFERENTES CONTEXTOS, ISSO POSSIBILITA A REUTILIZAÇÃO DOS OBJETOS EM DIFERENTES PARTES DO SISTEMA.

MELHORA NO DESEMPENHO

EM SITUAÇÕES EM QUE A CRIAÇÃO E A DESTRUIÇÃO DE MUITOS OBJETOS SÃO FREQUENTES. A REUTILIZAÇÃO DO ESTADO COMPARTILHADO EVITA A SOBRECARGA DE CRIAÇÃO DE NOVOS OBJETOS.

PONTOS NEGATIVOS

COMPLEXIDADE ADICIONAL

AO IMPLEMENTAR O PADRÃO FLYWEIGHT, A COMPLEXIDADE DO CÓDIGO FICA AINDA MAIOR, POIS REQUER A DIVISÃO DO ESTADO EM COMPARTILHADO E EXCLUSIVO

RESTRIÇÕES NO ESTADO COMPARTILHADO

O FLYWEIGHT PRESSUPÕE QUE PARTE DO ESTADO DO OBJETO PODE SER COMPARTILHADA ENTRE VÁRIAS INSTÂNCIAS. ISSO PODE IMPOR LIMITAÇÕES NO DESIGN, POIS NEM TODO O ESTADO PODE SER FACILMENTE COMPARTILHADO.

PERDA NO DESEMPENHO

EMBORA O FLYWEIGHT POSSA MELHORAR O DESEMPENHO EM DETERMINADAS SITUAÇÕES, ELE TAMBÉM PODE INTRODUZIR ALGUM OVERHEAD ADICIONAL, CAUSANDO UM PEQUENO CUSTO DE DESEPENHO