SERIALIZAÇÃO

PROFESSOR: LEANDRO MAIA
DISCIPLINA : PROG. ORIENTADA A OBJETOS

INTRODUÇÃO

SERIALIZAÇÃO

A serialização é um mecanismo poderoso em Java que permite transformar objetos em um formato que pode ser facilmente armazenado ou transmitido. Ao invés de o estado do objeto estar armazenado exclusivamente em memória, agora ele pode ser armazenado em armazenamento secundário.

INTRODUÇÃO

SERIALIZAÇÃO

A serialização permite "congelar" esse objeto, convertendo-o em uma sequência de bytes, que pode ser gravada em um arquivo, enviada pela rede ou armazenada em um banco de dados. Posteriormente, você pode "descongelar" essa sequência de bytes, recuperando o objeto original com todas as suas informações e estado.

SERIALIZAÇÃO

POR QUE SERIALIZAR?

A serialização é fundamental em diversas situações:

- Persistência de Dados: Permite salvar o estado de objetos em arquivos ou bancos de dados para uso posterior.

SERIALIZAÇÃO

POR QUE SERIALIZAR?

A serialização é fundamental em diversas situações:

- Comunicação entre Sistemas: Facilita a troca de objetos complexos entre diferentes sistemas ou componentes de software, através de redes ou outros meios de comunicação.

SERIALIZAÇÃO

POR QUE SERIALIZAR?

A serialização é fundamental em diversas situações:

- *Deep Copy*: Cria cópias profundas de objetos, preservando toda a sua estrutura interna e evitando problemas com referências compartilhadas.

SERIALIZAÇÃO

COMO FUNCIONA?

A serialização é realizada através da **interface java.io.Serializable**. Classes que implementam essa interface indicam que seus objetos podem ser serializados.

SERIALIZAÇÃO

COMO FUNCIONA?

Para serializar um objeto, você utiliza as classes **ObjectOutputStream** para escrever o objeto em um fluxo de saída e **ObjectInputStream** para ler o objeto de um fluxo de entrada.

SERIALIZAÇÃO

BENEFÍCIOS

- Simplicidade: A serialização é relativamente fácil de usar, exigindo apenas a implementação da interface Serializable.
- Flexibilidade: Permite personalizar o processo de serialização e desserialização para atender às suas necessidades específicas.

SERIALIZAÇÃO

BENEFÍCIOS

- Ampla aplicação: É amplamente utilizada em diversas áreas, como desenvolvimento web, aplicativos corporativos e jogos.

SERIALIZAÇÃO

DESAFIOS

- Compatibilidade: Alterações na estrutura de classes serializadas podem causar problemas de compatibilidade com versões anteriores.
- Segurança: A desserialização de dados não confiáveis pode ser uma vulnerabilidade de segurança.

A INTERFACE SERIALIZABLE

JAVA.IO.SERIALIZABLE

A interface Serializable é apenas uma interface marcadora, ou seja, ela não solicita a implementação de métodos.

Uma classe que implementa essa interface apenas sinaliza à JVM que ela pode ser serializada.

A INTERFACE SERIALIZABLE

JAVA.IO.SERIALIZABLE

A JVM faz todo o trabalho de transformar o objeto em um fluxo de bytes automaticamente.

O único requisito é que a classe e as superclasses devem implementar Serializable.

Os atributos que não devem ser serializados devem ser marcados como *transient*.

A INTERFACE SERIALIZABLE

JAVA.IO.SERIALIZABLE


```java
public class Produto implements Serializable {
    private String nome;
    private double preco;
    private transient int estoque; // Não será serializado
}
```

A CHAVE DE COMPATIBILIDADE

SERIALVERSIONUID

O SerialVersionUID é uma chave de compatibilidade que garante a integridade dos dados serializados.

A importância está no fato de permitir que a JVM verifique se a classe lida durante a desserialização é a mesma que foi serializada.

É essencial para manter a compatibilidade entre diferentes versões da sua aplicação.

A CHAVE DE COMPATIBILIDADE

SERIALVERSIONUID

Pode:

- Automática: pela JVM com valor padrão.
- Manualmente: pelo desenvolvedor, sendo o mais recomendado.

A CHAVE DE COMPATIBILIDADE

SERIALVERSIONUID


```java
public class Produto implements Serializable {

    private static final long serialVersionUID = 1L; // Valor definido manualmente

    private String nome;
    private double preco;
    private transient int estoque; // Não será serializado
}
```

ATRIBUTOS TRANSIENT

EXCLUSÃO DA SERIALIZAÇÃO

Atributos marcados com a palavra chave *transient* não são incluídos na serialização.

Esse marcador é particularmente importante porque nem tudo deve ou precisa ser serializado.

ATRIBUTOS TRANSIENT

EXCLUSÃO DA SERIALIZAÇÃO

Exemplos de uso:

- Para dados sensíveis (senhas).
- Para informações que não precisam ser persistidas (conexões de rede).
- Para atributos que não podem ser serializados (objetos de classes não-serializáveis).

ATRIBUTOS TRANSIENT

EXCLUSÃO DA SERIALIZAÇÃO


```
public class Usuario implements Serializable {
    private String nome;
    private transient String senha; // Não será serializado
}
```

CLASSES AUXILIARES

OBJECTOUTPUTSTREAM

A classe responsável por transformar um objeto serializável em um fluxo de bytes é a classe ObjectOutputStream. Essa classe possui métodos para serializar os tipos primitivos, além de String, mas nosso principal interesse é no método writeObject.

CLASSES AUXILIARES

OBJECTOUTPUTSTREAM


```java
public class Produto implements Serializable {

    private static final long serialVersionUID = 1L; // Valor definido manualmente

    private String nome;
    private double preco;
    private transient int estoque; // Não será serializado
}

  // Excerto de um trecho de uso do ObjectOutputStream
  public static void main(String[] args) {

    // ....

    FileOutputStream fileOut = new FileOutputStream("produto.ser");
    ObjectOutputStream out = new ObjectOutputStream(fileOut);
    out.writeObject(produto);
    out.close();
    fileOut.close();
  }
```

CLASSES AUXILIARES

OBJECTINPUTSTREAM

A classe responsável por transformar um fluxo de bytes em um objeto de volta é a classe ObjectInputStream. Assim como a ObjectOutputStream, possui métodos para desserializar os tipos primitivos e Strings, mas estamos interessados no método readObject.

CLASSES AUXILIARES

OBJECTINPUTSTREAM


```
public class Produto implements Serializable {

    private static final long serialVersionUID = 1L; // Valor definido manualmente

    private String nome;
    private double preco;
    private transient int estoque; // Não será serializado
}

  // Excerto de um trecho de uso do ObjectInputStream
  public static void main(String[] args) {

    // ....

    FileInputStream fileIn = new FileInputStream("produto.ser");
    ObjectInputStream in = new ObjectInputStream(fileIn);
    Produto produto = (Produto) in.readObject();
    in.close();
    fileIn.close();
  }
```

EXCESSÕES COMUNS

Nenhum processo de serialização / desserialização é garantido, principalmente se você **não controla** todas as classes do Sistema.

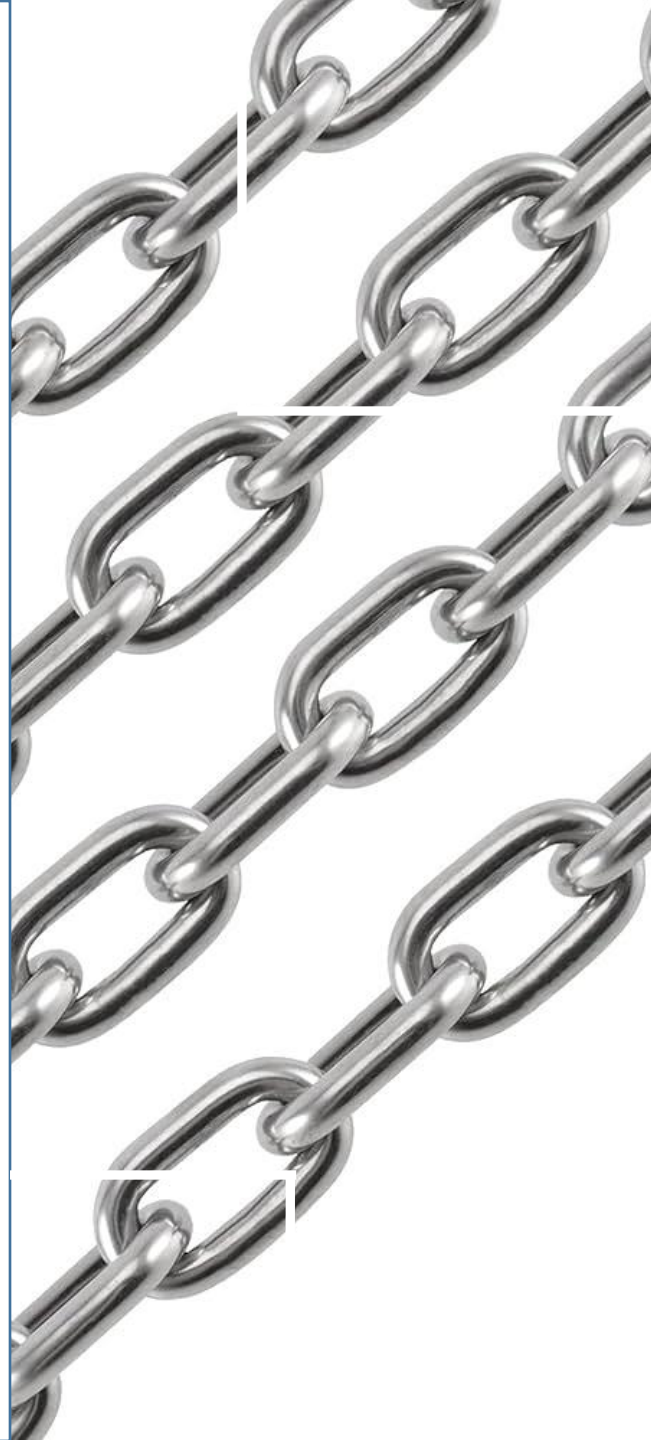
Por essa razão, são comuns as exceções **java.io.NotSerializableException** e **java.lang.ClassNotFoundException**.

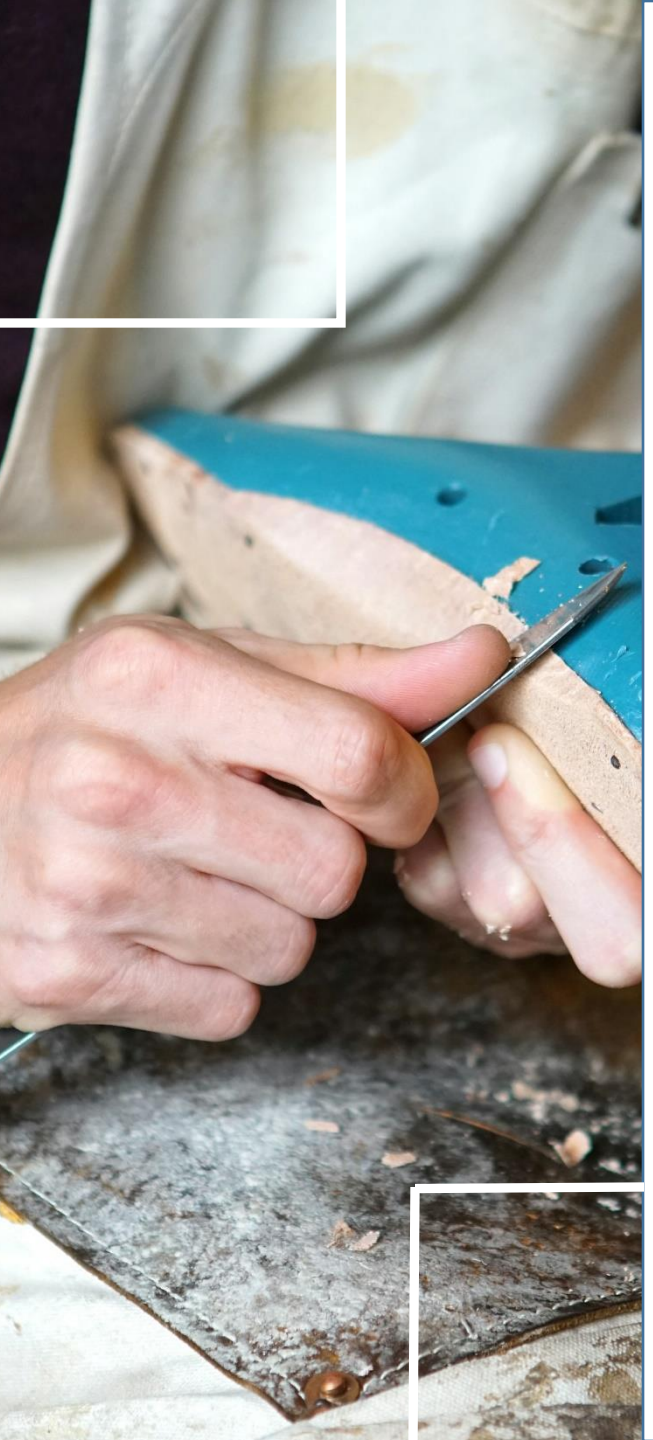
SERIALIZAÇÃO EM CADEIA

Objetos que referenciam outros objetos também os inclui durante a serialização.

Todo o processo continua sendo feito automaticamente pela JVM.

Um cuidado muito importante deve ser tomado com as referências circulares, pois elas causam a exceção *StackOverflowError*.



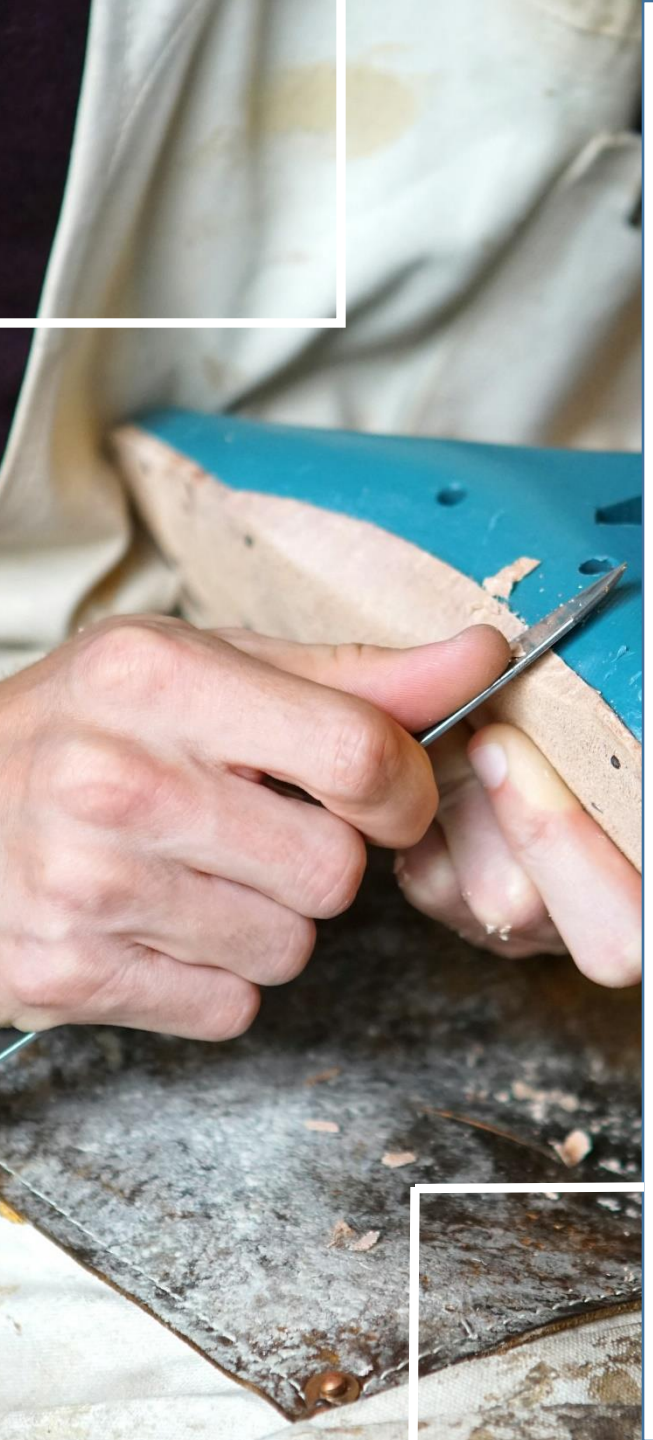


PERSONALIZANDO A SERIALIZAÇÃO

CONTROLE DA SERIALIZAÇÃO

Por que personalizar?

- Controlar o formato dos dados gravados.
- Otimizar o tamanho dos dados serializados.
- Lidar com objetos complexos ou com referências circulares.



PERSONALIZANDO A SERIALIZAÇÃO

CONTROLE DA SERIALIZAÇÃO

Como funciona?

Basicamente você vai precisar:

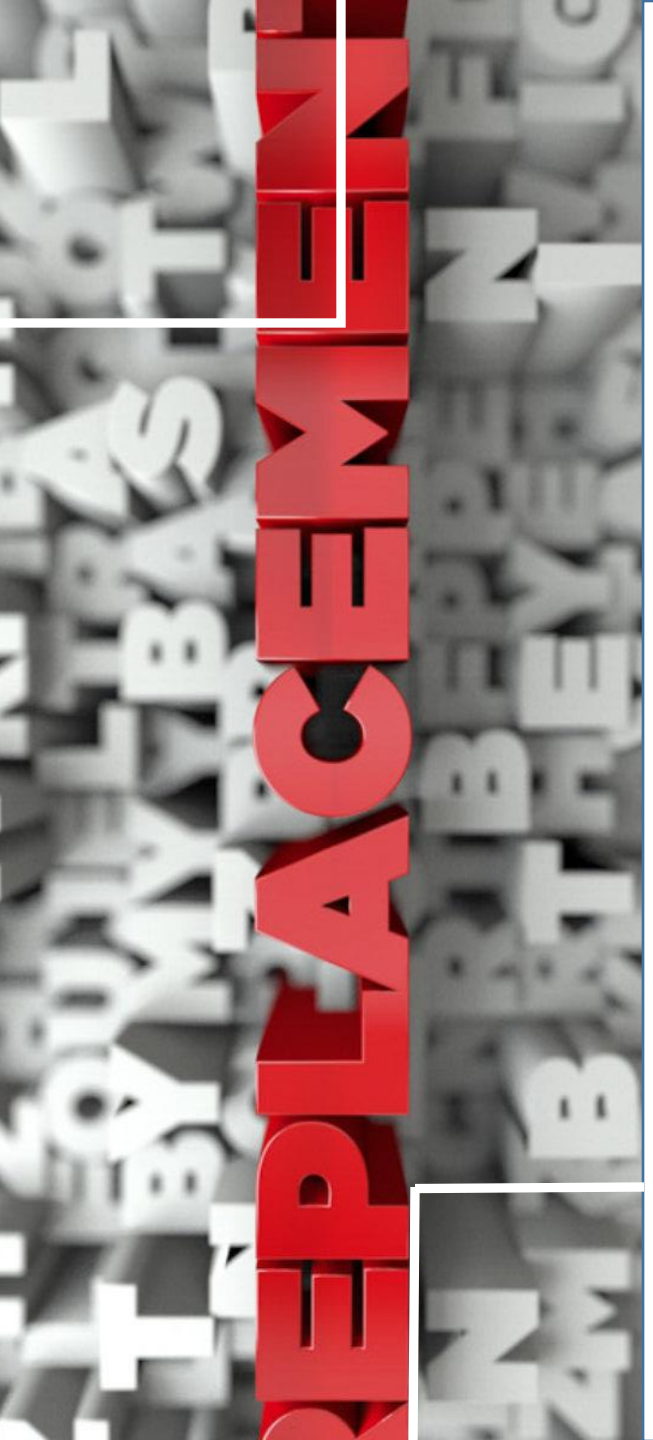
- Sobrescrever writeObject(ObjectOutputStream out) e readObject(ObjectInputStream in) na classe serializável.
- Escrever e ler os dados manualmente no fluxo de serialização.

PERSONALIZANDO A SERIALIZAÇÃO

CONTROLE DA SERIALIZAÇÃO


```
// Excerto de uma classe Serializable
private void writeObject(ObjectOutputStream out) throws IOException {
    // Serializa os atributos padrão
    out.defaultWriteObject();
    // Serializa um valor calculado
    out.writeInt(this.getValorCalculado());
}

private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {
    // Desserializa os atributos padrão
    in.defaultReadObject();
    // Desserializa o valor calculado
    int valorCalculado = in.readInt();
    // ... (reconstruir o estado do objeto)
}
```

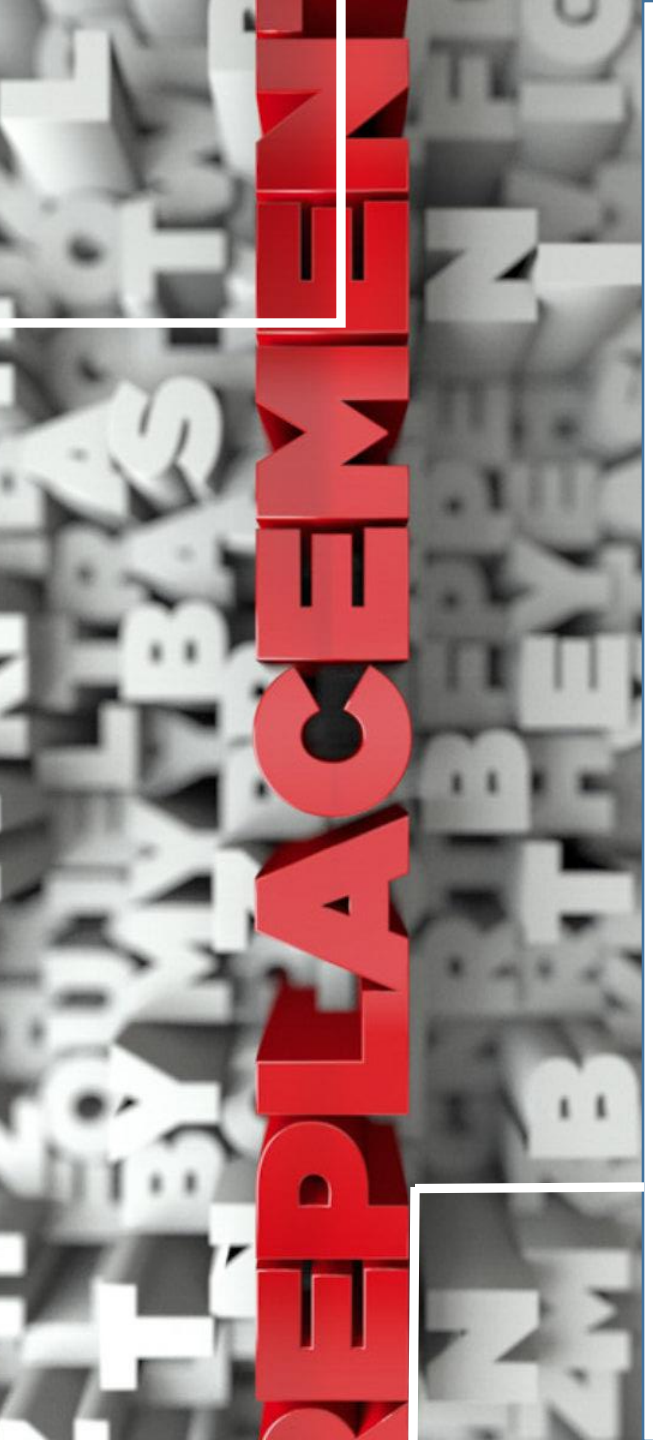


SUBSTITUINDO OBJETOS

WRITEREPLACE E READRESOLVE

Os métodos writeReplace e readResolve são utilizados para substituir um objeto por outro durante a serialização (writeReplace) ou desserialização (readResolve).

Isso permite ocultar a implementação real da classe e ao mesmo tempo garantir a consistência dos objetos.



SUBSTITUINDO OBJETOS

WRITEREPLACE E READRESOLVE

Geralmente esses métodos são utilizados no padrão Singleton ou para lidar com objetos muito complexos que precisam ser reconstruídos de uma forma muito específica.

SUBSTITUINDO OBJETOS

WRITEREPLACE E READRESOLVE


```
// Excerto de uma classe Serializable de padrão Singleton
private Object writeReplace() throws ObjectStreamException {
    // Retorna a instância única
    return Singleton.getInstance();
}

private Object readResolve() throws ObjectStreamException {
    // Retorna a instância única
    return Singleton.getInstance();
}
```

EXTERNALIZAÇÃO

CONTROLE TOTAL DA SERIALIZAÇÃO

A externalização é uma abordagem mais avançada de serialização que oferece total controle sobre o processo de transformar objetos em bytes e vice-versa.

EXTERNALIZAÇÃO

CONTROLE TOTAL DA SERIALIZAÇÃO

Diferente da interface Serializable, onde a JVM gerencia a serialização automaticamente, a interface Externalizable exige que os métodos *writeExternal* e *readExternal* sejam implementados para garantir explicitamente que você defina como cada atributo do objeto será serializado e desserializado.

EXTERNALIZAÇÃO

CONTROLE TOTAL DA SERIALIZAÇÃO

A interface java.io.Externalizable estende a interface Serializable, mas você precisa implementar:

- void writeExternal(ObjectOutput out)
- void readExternal(ObjectInput in)

EXTERNALIZAÇÃO

CONTROLE TOTAL DA SERIALIZAÇÃO

Quando usar?

- Controle Granular: Quando você precisa de controle total sobre o formato dos dados serializados. A Externalização permite que você escolha quais atributos serão serializados, em qual ordem e como eles serão representados no fluxo de bytes.

EXTERNALIZAÇÃO

CONTROLE TOTAL DA SERIALIZAÇÃO

Quando usar?

- Otimização: Em cenários onde o desempenho é crítico, a Externalização pode ser usada para otimizar o tamanho dos dados serializados, removendo informações desnecessárias ou usando formatos de dados mais compactos.

EXTERNALIZAÇÃO

CONTROLE TOTAL DA SERIALIZAÇÃO

Quando usar?

- Compatibilidade: A Externalização pode ajudar a manter a compatibilidade entre diferentes versões de uma classe, permitindo que você controle como os dados são lidos e escritos, mesmo que a estrutura da classe tenha mudado.

EXTERNALIZAÇÃO

CONTROLE TOTAL DA SERIALIZAÇÃO


```
public class Ponto implements Externalizable {
    private int x;
    private int y;

    // ... (construtores, getters, setters)

    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeInt(x);
        out.writeInt(y);
    }

    @Override
    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
        x = in.readInt();
        y = in.readInt();
    }
}
```

BOAS PRÁTICAS

COMPATIBILIDADE ENTRE VERSÕES

- Cuidado ao modificar classes serializadas.
- Uso do serialVersionUID para garantir compatibilidade.
- Estratégias para lidar com incompatibilidades.

BOAS PRÁTICAS

SEGURANÇA NA SERIALIZAÇÃO

- Vulnerabilidades: Injeção de objetos maliciosos.
- Mitigação:
  - Validar dados desserializados.
  - Usar filtros de desserialização.
  - Desabilitar a desserialização de classes não confiáveis.

BOAS PRÁTICAS

ALTERNATIVAS À SERIALIZAÇÃO

- JSON: Mais leve e interoperável.
- Protocolo Buffers: Eficiente para comunicação entre sistemas.
- XML: Mais verboso, mas amplamente suportado.
- Escolha da tecnologia ideal para cada cenário.

BOAS PRÁTICAS

DESEMPENHO DA SERIALIZAÇÃO

- Serialização pode ser custosa em termos de tempo e recursos.
- Considerar alternativas mais leves em cenários críticos de desempenho.
- Otimizações:
  - Serialização seletiva de atributos.
  - Compressão de dados serializados.