

Estrutura de Dados II

Ciência da Computação

Prof. André Kishimoto
2024

1

TAD e Estrutura de Dados

Tipos abstratos

Abstração

- Algo que existe como ideia ou conceito, mas talvez não concretizado.
- “Usamos um objeto sem saber o seu mecanismo interno.” (ex. carro, para a população em geral).
- Refinar um sistema complicado para suas partes fundamentais, descrevendo essas partes em uma linguagem simples e precisa.
 - Dar nomes e
 - Explicar suas funcionalidades.

Tipos abstratos

Ao aplicarmos o conceito de abstração em um tipo de dado, começamos a definir tipos abstratos.

Tipo Abstrato de Dado (TAD) ou Abstract Data Type (ADT).

- Baseado em um tipo de dado,
 - Especifica **o que** cada operação realiza.
 - *Não* especifica *como* a operação realiza sua tarefa.

Em outras palavras... Quais são as funções necessárias para manipular os dados do problema e o que cada função deve fazer?

Tipos abstratos

Um TAD pode ser visto como um contrato entre três pessoas:

- Projetista define os dados e as operações (TAD).
- Programador(a) codifica as funções que realizam as operações.
- Qualquer pessoa usuária da versão concreta (implementada) do TAD sabe quais funções estão disponíveis para uso e o que fazem.

Tipos abstratos

Declarações de pré-condição e pós-condição definem o antes e o depois de uma operação.

- Pré-condição:
 - Indica o que deve ser verdadeiro antes de uma operação ser executada.
 - Garantir que a pré-condição seja válida é responsabilidade da pessoa usuária do TAD, mas quem implementa o TAD pode usar programação defensiva e verificar se os valores são válidos.
- Pós-condição:
 - Indica o que será verdadeiro após uma operação ser executada.
 - Quem implementa o TAD deve garantir que a pós-condição seja válida.

Tipos abstratos

Principais benefícios dos TADs

- Encapsulamento
 - O código das operações fica “escondido” da pessoa usuária do TAD.
 - Acesso via interface do TAD.
- Abstração
 - Pessoa usuária do TAD não precisa saber como TAD foi implementado.
- Reuso
 - TADs podem ser reusados em diferentes projetos e contextos.

Tipos abstratos

Principais benefícios dos TADs

- Manutenção do código
 - Manutenção/melhorias no código interno do TAD, sem afetar usuários.
- Padronização
 - TAD bem projetado segue algum padrão para as operações e a interface.
 - Uso do TAD fica consistente, podendo reduzir erros de programação.
- Segurança
 - Restringir acesso e manipulação direta dos dados.
 - Melhora segurança do código e reduz problemas com os dados.

TAD e Estrutura de Dados

Uma estrutura de dados é uma maneira de definir como os dados são organizados, armazenados e manipulados por um programa de computador.

TAD = abstração.

Estrutura de dados = implementação concreta de um TAD.

TAD (“o que”) vs. Estrutura de dados (“como”)

2

Classes e Objetos

Classes e Objetos

Conceito fundamental do paradigma de programação orientada a objetos.

Programação Orientada a Objetos (POO) ou Object-Oriented Programming (OOP).

Paradigma baseado no conceito de “Objetos”.

Objetos podem corresponder a coisas e ideias encontradas no mundo real (pessoas, animais, veículos, figuras geométricas, etc.).

Software composto por vários objetos que possuem relações entre si para resolver problemas.

Classes

- Classes são os “blocos de construção” em Programação Orientada a Objetos (POO).
- A definição de uma classe não cria nenhum objeto.
- É um modelo que define propriedades e comportamentos de um objeto.
- Membros de dados e membros que são funções.

```
class <NomeDaClasse>  
{  
    // atributos (variáveis)  
    // métodos (funções com ou sem retorno)  
};
```

Classes

- A classe `MyPositiveInt` do próximo slide pode ser usada para armazenar um número do tipo `int` que só aceita números inteiros positivos.
- Caso um número fornecido em `setValue()` seja negativo, o número será convertido para positivo, usando o método `abs()` (valor absoluto).

Classes

```
public class MyPositiveInt {  
  
    private int value;  
  
    MyPositiveInt() {  
        this(0);  
    }  
  
    MyPositiveInt(int value) {  
        setValue(value);  
    }  
  
    public int getValue() {  
        return value;  
    }  
}
```

```
    public void setValue(int value) {  
        this.value = abs(value);  
    }  
  
    private int abs(int value) {  
        return value < 0  
            ? -value  
            : value;  
    }  
  
    @Override  
    public String toString() {  
        return Integer.toString(value);  
    }  
}
```

Objetos

Objetos possuem

- **Identificador:** Um nome exclusivo que identifica o objeto e permite que um objeto interaja com outros objetos.
- **Estado:** Representado pelos atributos (propriedades) do objeto.
- **Comportamento:** O que o objeto pode realizar.

Objetos

Objetos possuem

- Identificador: **Nome** da classe (novo tipo de dado) e do objeto (variável do tipo da classe).
- Estado: **Atributos** (variáveis) da classe.
- Comportamento: **Métodos** (funções) da classe.

Objetos

- Um objeto é uma instância (implementação) de uma classe.
- Quando um objeto de uma classe é criado, dizemos que a classe foi instanciada.
- Toda instância tem os mesmos estados e comportamentos (atributos e métodos) da classe.
 - Isto é, a mesma estrutura/interface.
- Porém, cada instância da classe (objeto) tem seu próprio estado.
 - Isto é, valores de atributos próprios (valores específicos).

Método construtor da classe

- O que acontece quando um objeto é criado?
- O método construtor da classe é executado.
 - Como se fosse a `main()` de um programa Java.
- O construtor é um método especial.
 - [Geralmente] usado para iniciar/definir valores iniciais/padrões dos objetos.
 - Possui o mesmo nome da classe.
 - Não tem retorno (nem mesmo `void`).
 - Executado quando objeto é criado.
 - Se não implementamos o método construtor, a linguagem Java cria um construtor padrão.

Objetos

```
// Criação de objetos
```

```
<NomeDaClasse> <nomeDaVariavel>;
```

```
MyPositiveInt obj = new MyPositiveInt();
```

```
MyPositiveInt arr[] = new MyPositiveInt[5];
```

```
for (int i = 0; i < arr.length; ++i) {
```

```
    arr[i] = new MyPositiveInt(-i);
```

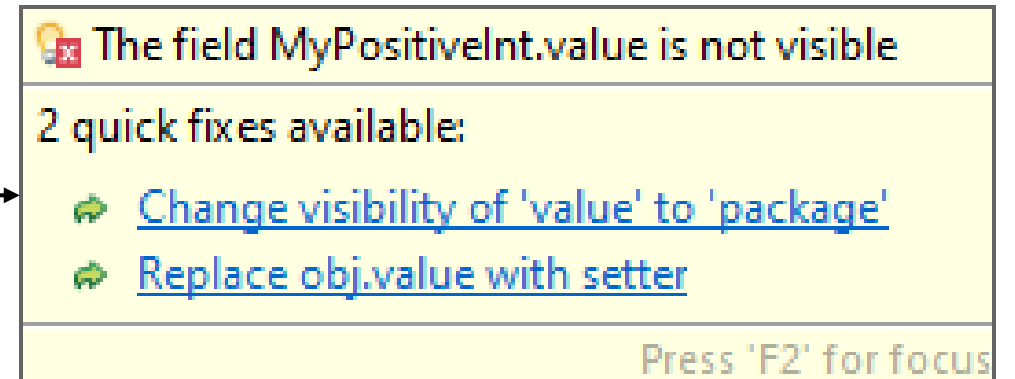
```
}
```

Encapsulamento

- Ocultar membros (ou tornar visíveis).
- Escondemos parte da classe, de forma que o que está “escondido” não é acessível para códigos fora da classe.
- Dois modificadores de acesso comuns: **private** e **public**.
- Podemos pensar em exemplos da vida (desconsiderando alguns detalhes...):
 - Acesso ao interior de um prédio: a portaria define quem pode e não pode entrar.
 - Pegar um produto que está atrás do balcão de uma loja. Ao invés do cliente pegar o produto diretamente, algo restrito/privado da loja, o cliente deve pedir para o vendedor pegar o produto.

Encapsulamento

```
public class Review {  
  
    public static void main(String[] args) {  
  
        MyPositiveInt obj = new MyPositiveInt();  
  
        obj.value = -10;  
  
        obj.setValue(-10);  
  
    }  
}
```

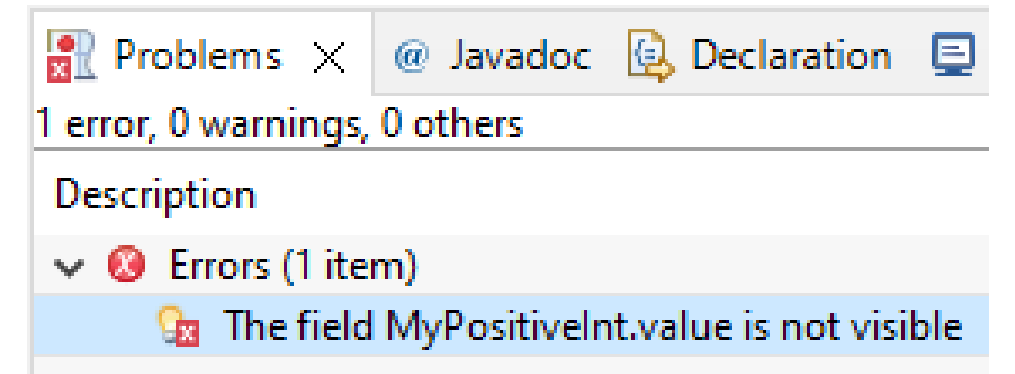


The field `MyPositiveInt.value` is not visible

2 quick fixes available:

- [Change visibility of 'value' to 'package'](#)
- [Replace obj.value with setter](#)

Press 'F2' for focus



Problems × @ Javadoc Declaration

1 error, 0 warnings, 0 others

Description

Errors (1 item)

The field `MyPositiveInt.value` is not visible