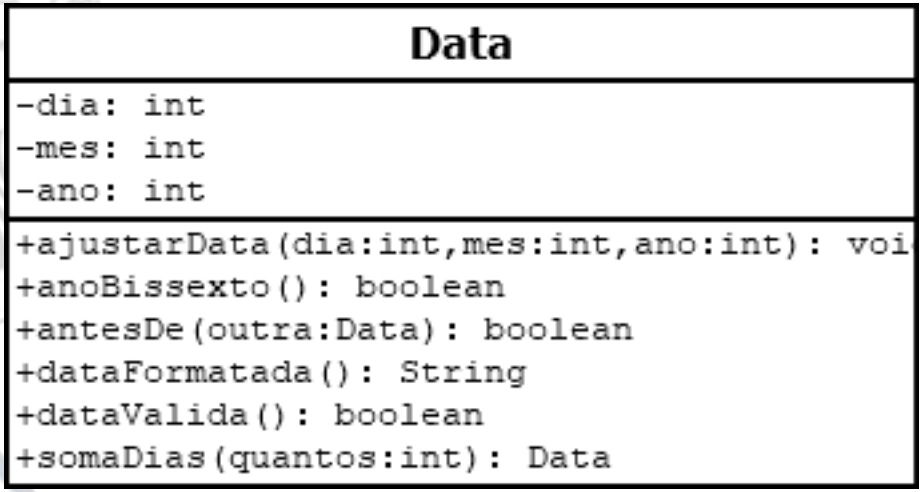
Programação Modular

Fundamentos de POO: construtores e destrutores

Prof Dr Johnatan Oliveira

Classes, objetos e estados

- Suponha a existência de uma classe *Data:*
  - armazena datas do calendário;
  - permite algumas operações com elas.



Classes e objetos

```
//instância da Classe Data
Data dataUm = new Data();
```

Classes e objetos

```
//instância da Classe Data
Data dataUm = new Data();

boolean bissexto = dataUm.anoBissexto();
System.out.println("O ano é bissexto? " + bissexto);
```

Classes e objetos

```
//instância da Classe Data
Data dataUm = new Data();

boolean bissexto = dataUm.anoBissexto();
System.out.println("O ano é bissexto? " + bissexto);
```

Quanto vale a data neste momento?

Classes e objetos

```
//instância da Classe Data
Data dataUm = new Data();

dataUm.ajustarData(14,08,2022);
boolean bissexto = dataUm.anoBissexto();
System.out.println("O ano é bissexto? " + bissexto);
```

Classes e objetos

```
//instância da Classe Data
Data dataUm = new Data();

dataUm.ajustarData(14,08,2022);
boolean bissexto = dataUm.anoBissexto();
System.out.println("O ano é bissexto? " + bissexto);
```

OK, porém....

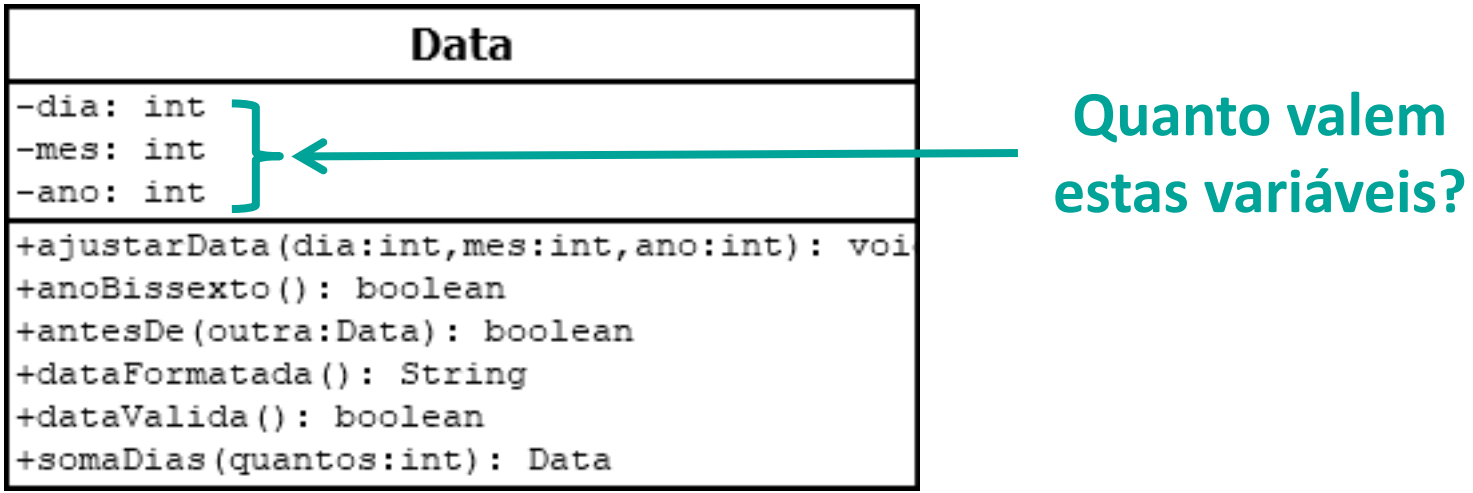
Tedioso!

E se alguém esquecer?

Classes e objetos


```
//instância da Classe Data
Data dataUm = new Data();
```

- O que acontece com o objeto em sua criação?



Construtores

- Construtores são usados para criar e iniciar objetos com valores diferentes do padrão:
  - Em geral, possuem o mesmo nome da classe.
  - Não possuem valores de retorno.

- Uma classe pode ter de um (oculto) a muitos construtores.

Boas razões para usar construtores

- Estado inicial padrão pode não ser aceitável;
- Fornecer um estado inicial consistente é conveniente ao criar objetos;
- Construir um objeto aos poucos é desgastante;
- Um construtor pode ser privado, restringindo assim o uso de sua classe.

Construtor padrão (*default*)

- Utilizados pela linguagem quando não há método construtor implementado.

- Geralmente inicializa todos os atributos com *null*, *false* ou zero.

```
Data dataUm = new Data();
```

Construtor padrão (*default*)

- Utilizados pela linguagem quando não há método construtor implementado.
- Geralmente inicializa todos os atributos com *null*, *false* ou zero.
- Mas... Pode simplesmente não inicializar nada.

Construtor padrão (*default*)

- O construtor padrão pode ser inseguro ou levar a estados inválidos de objetos.

```
Data dataUm = new Data();
boolean ok = dataUm.dataValida();
```

Construtor padrão (*default*)

```
Data dataUm = new Data();
boolean ok = dataUm.dataValida();
```

Temos uma data válida?
Ela deveria ser válida?

Construtor próprio

- Implementa-se um construtor para se ter certeza de que as regras definidas para a classe são seguidas.

```
Data dataUm = new Data();
bool ok = dataUm.dataValida();
```

Construtor próprio

```
class Data{
    private int dia;
    private int mes;
    private int ano;
    public Data(){
      this.dia = 1;
      this.mes = 1;
      this.ano = 1900;
    };
...
}
```

```
Data dataUm = new Data();
System.out.println(dataUm.dataFormatada());
```

Construtores

- Procure criar seus construtores próprios.
  - Inicialização correta e completa.

- Na maioria das linguagens OO, uma classe pode ter vários construtores, variando-se a quantidade ou tipos de parâmetros.

Múltiplos construtores

```
public Data(){
    this.dia = 1;
    this.mes = 1;
    this.ano = 1900;
};
```

```
public Data(int d, int m, int a){
    this.dia = d;
    this.mes = m;
    this.ano = a;
    if(!this.dataValida()){
        this.dia = 1;
        this.mes = 1;
        this.ano = 1900;
    }
};
```

Múltiplos construtores

- Múltiplos construtores podem levar a inconsistências ou a código duplicado.

Múltiplos construtores

```
public Data(){
    this.dia = 1;
    this.mes = 1;
    this.ano = 1900;
};
```

```
public Data(int d, int m, int a){
    this.dia = d;
    this.mes = m;
    this.ano = a;
    if(!this.dataValida()){
        this.dia = 1;
        this.mes = 1;
        this.ano = 1900;
    }
};
```

Múltiplos construtores

```
public Data(){
    this.dia = 1;
    this.mes = 1;
    this.ano = 1900;
};
```

```
public Data(int d, int m, int a){
    this.dia = d;
    this.mes = m;
    this.ano = a;
    if(!this.dataValida()){
        this.dia = 1;
        this.mes = 1;
        this.ano = 1900;
    }
};
```

Método inicializador

- Um artifício comum é criar um método inicializador, o qual pode ser chamado pelos diferentes construtores.

Método inicializador

```
private void init(int dia, int mes, int ano){
    this.dia = dia;
    this.mes = mes;
    this.ano = ano;
    if(!this.dataValida()){
       this.dia = 1;
       this.mes = 1;
       this.ano = 1900;
    }
}
```

Inicializador e constutores

```
//O inicializador é utilizado pelos construtores

public Data(){
   init(1,1,1900);
}
public Data(int dia, int mes){
   init(dia,mes,2022);
}
public Data(int dia, int mes, int ano){
   init(dia,mes,ano);
}
```

Inicializador

- Outra vantagem: *Estado oculto!*
  - Mudanças na regra de inicialização têm impacto apenas no método inicializador.

Múltiplos construtores

- Mas: cuidado!!! Se você criou um construtor para a classe, o compilador **não** cria um construtor padrão.

```
class Data{
    public Data(int dia, int mes, int ano){
        this.init(d,m,a);
    };
}
```

Múltiplos construtores

```
class Data{
     public Data(int dia, int mes, int ano){
         this.init(d,m,a);
     };
}
```

```
Data dtNascimento = new Data();
```

Erro!

Mais exemplos de construtores JAVA


```
public class Pessoa {
    String nome;
    int idade;
public Pessoa(String nome, int idade) {// Construtor
        this.nome = nome;
        this.idade = idade;
    }
    public static void main(String[] args) {
        Pessoa pessoa1 = new Pessoa("João", 25);
        Pessoa pessoa2 = new Pessoa("Maria", 30);
}
}
```

Mais exemplos de construtores JAVA /2


```
public class Carro {
    String marca;
    String modelo;
public Carro() {// Construtor padrão
        marca = "Desconhecida";
        modelo = "Desconhecido";
    }
    public static void main(String[] args) {
        Carro carroPadrao = new Carro();
        System.out.println("Carro Padrão: " + carroPadrao.marca + " " +
carroPadrao.modelo);
    }
```

Mais exemplos de construtores JAVA /Sobrecarga


```java
public class Livro {
    String titulo;
    String autor;
    int anoPublicacao;

    // Construtor com sobrecarga
    public Livro(String titulo, String autor, int anoPublicacao) {
        this.titulo = titulo;
        this.autor = autor;
        this.anoPublicacao = anoPublicacao;
    }

    // Construtor alternativo sem o ano de publicação
    public Livro(String titulo, String autor) {
        this.titulo = titulo;
        this.autor = autor;
        this.anoPublicacao = -1; // Valor padrão para ano desconhecido
    }

    public static void main(String[] args) {
        Livro livro1 = new Livro("Dom Casmurro", "Machado de Assis", 1899);
        Livro livro2 = new Livro("1984", "George Orwell");

        System.out.println("Livro 1: " + livro1.titulo + " (" + livro1.anoPublicacao + ")");
        System.out.println("Livro 2: " + livro2.titulo + " (" + livro2.anoPublicacao + ")");
    }

}
```

DESTRUTORES

Destrutores (*destructors)*

- Métodos especiais invocados quando um objeto é finalizado (capturado pelo coletor de lixo).
  - Só há um destrutor por classe.
  - Um destrutor não tem parâmetros.
  - Não deve ser chamado diretamente.

Destrutores (*destructors)*

- Objetivo: Liberação de recursos usados pelo objeto

  - Ex: memória
    arquivos
    conexões de rede, bancos de dados..

Destrutores (C++)

```
~nomeDaClasse(){
      //seu código aqui
 }
```

Coletor de lixo

- Processo que libera automaticamente memória que não está sendo mais utilizada.
  - Eliminam a necessidade de se desalocar memória explicitamente;
  - Eliminam o vazamento de memória;
  - Eliminam referências pendentes (*dangling pointer*).

Java e coletor de lixo

- Linguagem Java:
  - não permite acesso direto à memória;
  - Não possui operadores de liberação de memória;
    - C, C++: *free*, *delete*
  - Possui coletor de lixo ***(garbage collector)***.

Java e coletor de lixo

- Um objeto é elegível para coleta de lixo quando:
  - não é mais acessado por nenhuma referência;
  - referencia um outro objeto que também o referencia, formando um ciclo único e isolado.
- O coletor de lixo é autônomo.
- Método ***System.gc()***

Destrutores (Java)

- São chamados automaticamente pelo coletor de lixo.
  - Execução do método ***finalize()*** da classe.
  - Finalize: Método obsoleto (***deprecated***).

- Java 9: Classe ***Cleaner*** e método ***clean()***;

Para Ficar mais claro

- Em Java, você não precisa escrever explicitamente um destrutor para liberar memória. O sistema de gerenciamento de memória Java utiliza o coletor de lixo (garbage collector) para automaticamente liberar a memória ocupada por objetos que não estão mais sendo referenciados.

Para Ficar mais claro

- O coletor de lixo identifica objetos que não possuem mais referências válidas (isto é, nenhum objeto no programa pode acessá-los) e libera a memória que esses objetos ocupavam. Isso ocorre em segundo plano e é uma característica chave do Java para evitar vazamentos de memória.

Para Ficar mais claro

- Em resumo, os construtores são usados para inicializar objetos, enquanto o sistema de gerenciamento de memória em Java, incluindo o coletor de lixo, cuida da liberação automática de memória para objetos não utilizados.

C# e Destrutores

- Há uma classe estática GC em C#
  - *Chamada manual da coleta.*

- Classes em C# podem implementar a interface ***IDisposable***
  - Método ***Dispose()***
  - Liberação de recursos sem destruir o objeto