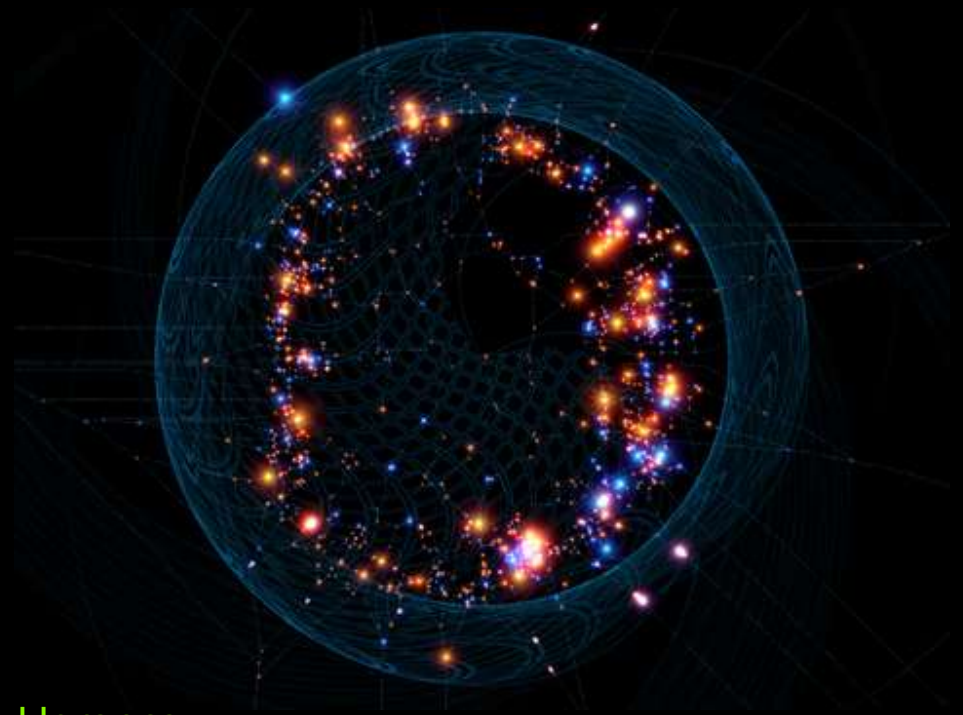
Deloitte.



Tecnologias de mercado - Herança

- Janeiro2022

Exercício

Animais

- Definir as classes Cao, Ovelha, Avestruz e Hamster

- O método correr escreve para o Standard output "Estou a correr com X patas"

Cao
-noDePatas:int -comida:String -nome:String -noOssosEnterrados:int
+Cao(noDePatas:int, comida:String, nome:String) +getNoDePatas():int +setNoDePatas(noDePatas):void +getComida():String +setComida(comida:String):void +correr():void +incOssosEnterrados():void +getOssosEnterrados():int +toString():String

Ovelha
-noDePatas:int -comida:String -nome:String -temLa:boolean
+Ovelha(noDePatas:int, comida:String, nome:String) +getNoDePatas():int +setNoDePatas(noDePatas):void +getComida():String +setComida(comida:String):void +correr():void +getTemLa():boolean +setTemLa(temLa:boolean):void +toString():String

Avestruz
-noDePatas:int -comida:String -nome:String
+Avestruz(noDePatas:int, comida:String, nome:String) +getNoDePatas():int +setNoDePatas(noDePatas):void +getComida():String +setComida(comida:String):void +correr():void +voar():void +toString():String

Hamster
-noDePatas:int -comida:String -nome:String
+Hamster(noDePatas:int, comida:String, nome:String) +getNoDePatas():int +setNoDePatas(noDePatas):void +getComida():String +setComida(comida:String):void +correr():void +toString():String

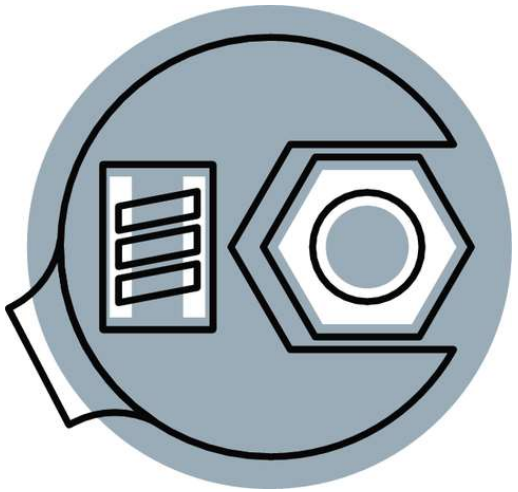
- O método voar escreve "As avestruzes não voam"

- O método correr da classe Hamster escreve para o standard output "Estou a correr com X patas, na minha rodinha"

Exercicio

Animais - problema

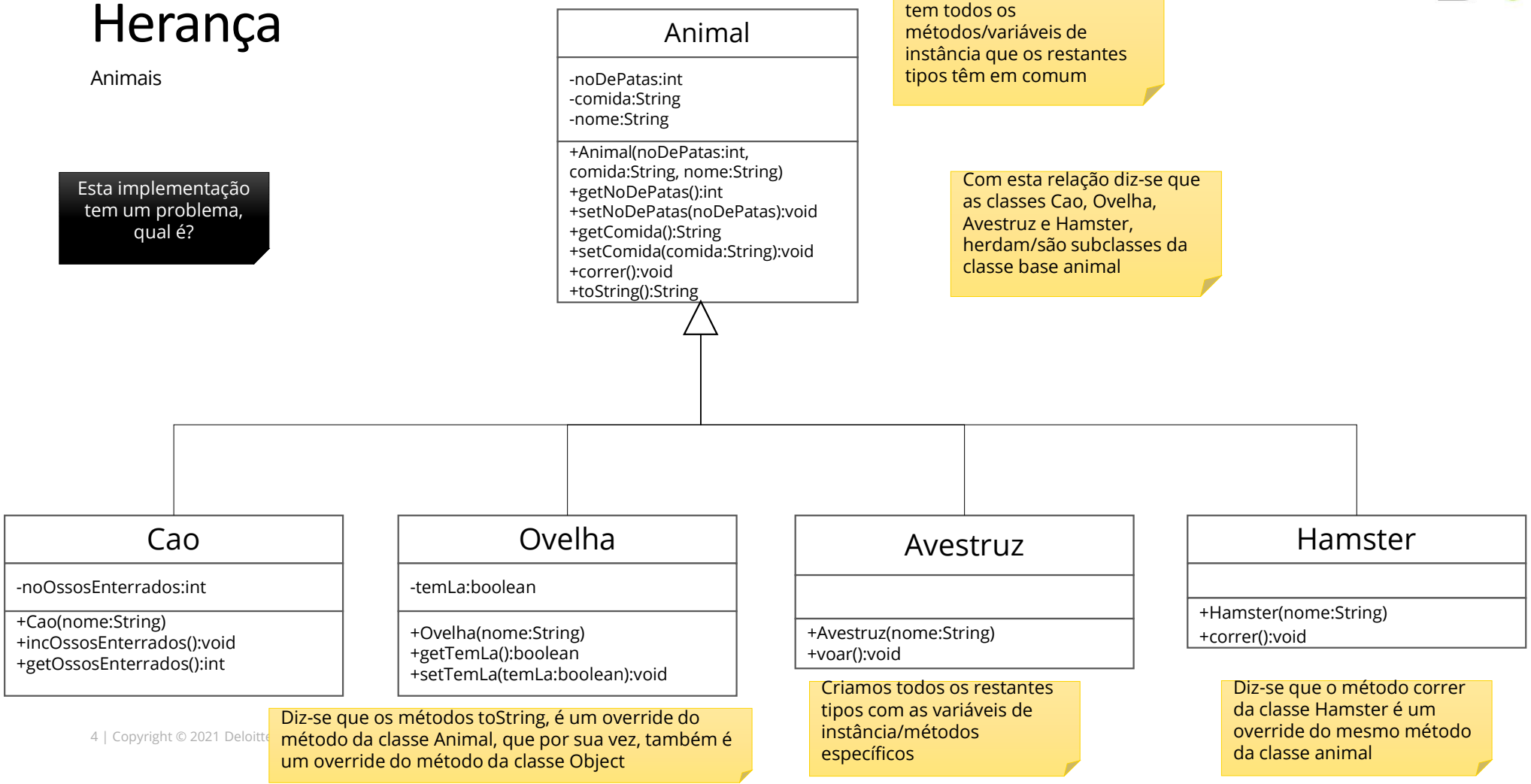
- Difícil manutenção
  - Se for necessário adicionar mais alguma informação a cada um dos tipos (exemplo: um identificador) é necessário adicionar em todas as classes;
  - Há muito código repetido, desde a definição de variáveis aos getters, setters e ao próprio método correr;



Herança

Animais

Esta implementação tem um problema, qual é?



Criamos o tipo Animal, que tem todos os métodos/variáveis de instância que os restantes tipos têm em comum

Com esta relação diz-se que as classes Cao, Ovelha, Avestruz e Hamster, herdam/são subclasses da classe base animal

Diz-se que os métodos toString, é um override do método da classe Animal, que por sua vez, também é um override do método da classe Object

Criamos todos os restantes tipos com as variáveis de instância/métodos específicos

Diz-se que o método correr da classe Hamster é um override do mesmo método da classe animal

Herança

Não esquecer

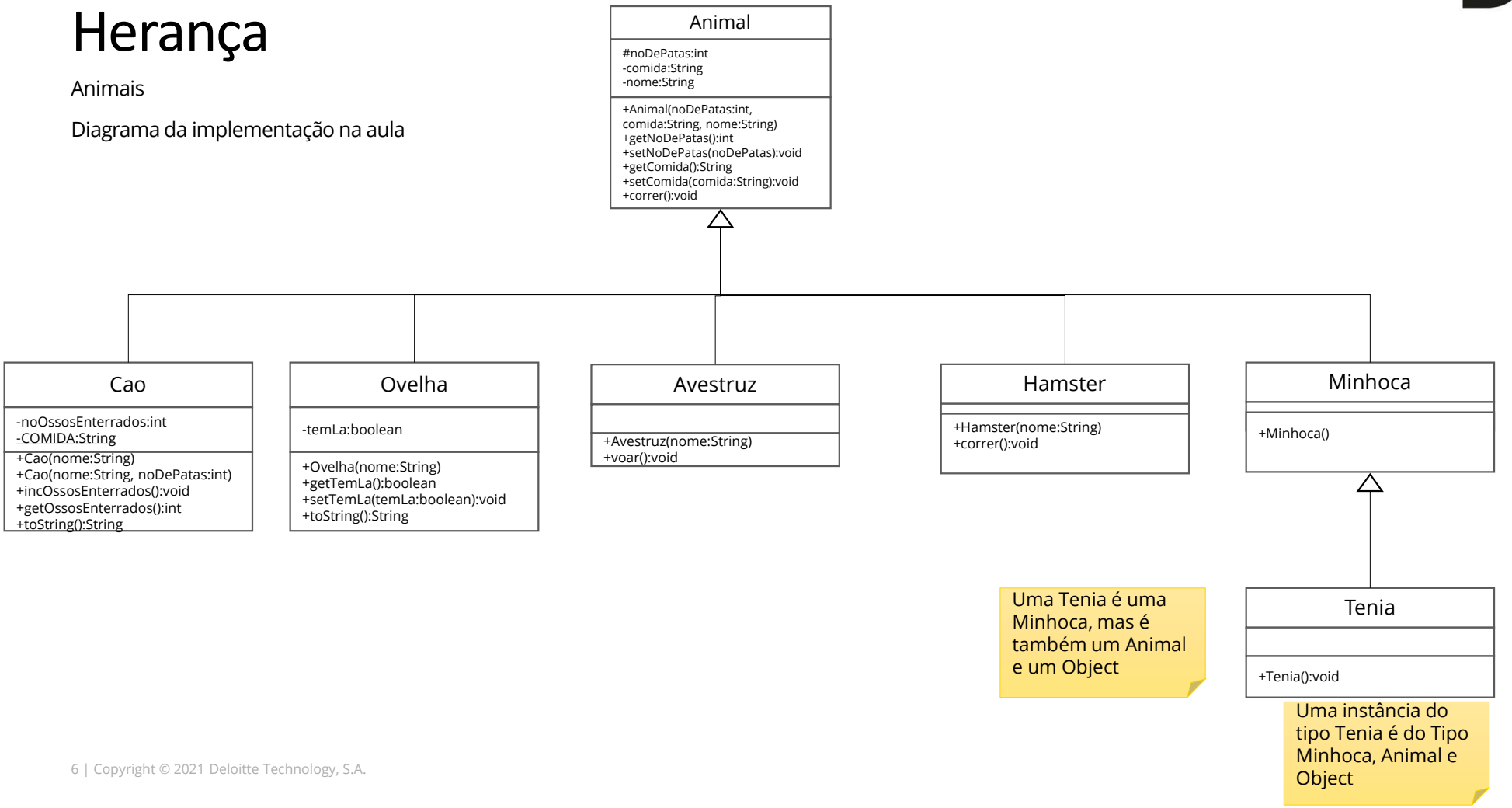
- Para uma classe herdar de outra, usa-se a *keyword* **extends** – motivo pelo qual também se pode dizer que, por exemplo, a classe Cao estende a classe Animal;
- Uma classe pode apenas estender uma única classe;
- Enquanto que na agregação se diz que uma classe **tem** uma ou mais instâncias de outras, ou que uma classe **usa** outra. Na Herança diz-se que uma classe **é** outra. Por exemplo, uma Ovelha **é** um Animal;
- As instâncias de subclasses são do tipo da classe que o define, mas também do tipo das classes base – uma instância de Avestruz é do tipo avestruz mas também é do tipo Animal e:
    - Como **todas** as classes em Java derivam direta ou indiretamente de **Object,** diz-se que uma instância de Avestruz é do tipo Avestruz e Animal mas também do tipo **Object;**

- Diz-se que um método é um override de outro quando, este tem o mesmo tipo de retorno e assinatura de um método herdado, isto é, **redefine comportamento da classe derivada** (por exemplo, o que já fizemos com o método toString());
- Não confundir Overriding com Overloading:
    - Ao fazer Override, o tipo de retorno, método e parâmetros são os mesmos definidos na classe base;
    - Ao fazer Overload, o tipo de retorno e nome do método são iguais mas o número de parâmetros muda – pode ser dentro da mesma classe ou na classe base;
- Para chamar o construtor ou um método da classe base, usa-se a *keyword* **super** para nos referirmos explicitamente à classe base – motivo pelo qual também se pode chamar à classe base "***super class***";
- Para chamar um construtor dentro de outro construtor (overload) deve-se usar a *keyword* **this;**
- Ao colocar o *modifier* **final** num método ou numa classe, estes não podem ser overriden ou estendidos respetivamente - gera erro de compilação;

Herança

Animais

Diagrama da implementação na aula



Herança

Rever modificadores de acesso

Variáveis de instância e métodos privados de uma classe base nunca estarão acessíveis nas classes base:

- Na classe Cao, não há acesso aos membros (variáveis e métodos de instância) privados da classe base, mas pode aceder às variáveis de instância través dos métodos públicos;
- Para disponibilizar variáveis de instância e métodos para as subclasses sem as tornar públicas, usa-se a *keyword* **protected;**

Modificador de acesso	Dentro da classe	Dentro do package	Fora do package	Fora do package mas numa subclasse
Private	Y	N	N	N
Default ou package-private	Y	Y	N	N
Protected	Y	Y	N	Y
Public	Y	Y	Y	Y

Herança

Exercício

- Considere uma empresa, que é definida por:
  - Nome;
  - Capital social (double);
  - Colaboradores;
- Cada colaborador é definido por:
  - Número de empregado;
  - Nome;
  - Salário (double);
  - Tarefas;
- As Tarefas são definidas por:
  - Tempo execução em minutos (int);
  - Descrição;
- Considere também que a empresa tem 3 tipos de colaborador:
  - Técnico TI;
  - Assistente executiva;
  - Gestor;
- Cada colaborador deve ter gerar uma lista de tarefas aleatória, cuja soma do tempo seja de 8 horas – cada tarefa deve ser no mínimo de 15 minutos;
- As tarefas possíveis para os colaboradores são:
  - **Técnico TI**: Análise Incidentes, Programar e Esclarecimentos;
  - **Assistente Executiva**: Apoio à faturação, Gestão de agenda, Reuniões e Gestão de Engagement;
  - **Gestor**: Gestão de equipa e Reuniões;
- Crie uma aplicação que mostre o relatório de pessoal de duas empresas, em que apresente a lista de colaboradores, com a sua lista de tarefas e duração.