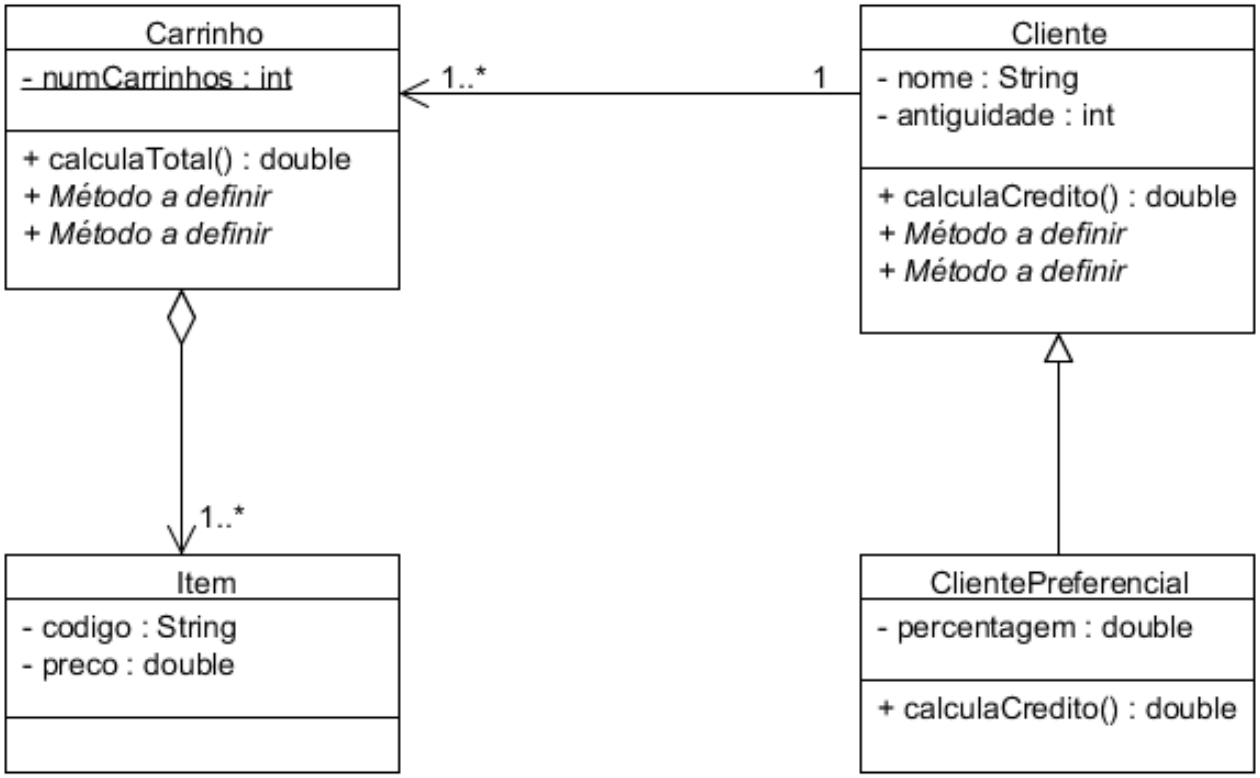
UNIVERSIDADE da MADEIRA

RESOLUÇÃO DE EXAME ANTERIOR I

Mónica Cameirão
monica.cameirao@staff.uma.pt

Grupo I - 1

Considere o seguinte diagrama de classes UML, que descreve uma aplicação para gestão de um supermercado. Neste diagrama **foram omitidos** eventuais construtores, métodos seletores e modificadores, e métodos **toString**.



1. Indique o nome e descreva cada uma das relações existentes neste diagrama, e o que estas implicam.

Grupo I - 2

2. Defina em JAVA a classe **Cliente**. Os atributos **nome** (nome do cliente) e **antiguidade** (anos como cliente) têm de ser fornecidos ao instanciar um novo objeto desta classe. Inclua os métodos seletores. O método **calculaCredito** calcula o crédito permitido a um dado cliente, onde o valor do crédito é obtido multiplicando os anos de cliente por um valor base de 500 euros; defina uma constante para este valor base. Tenha em atenção que tem de implementar a relação com a classe **Carrinho**. O primeiro método a definir deve permitir registar um novo carrinho. O segundo método a definir é o método **equals** (faça *overriding*) que verifica a igualdade entre dois clientes em termos de nome. Inclua um método **toString** que imprime o nome e a antiguidade do cliente e o número de carrinhos desse cliente.

Grupo I - 2


```java
import java.util.ArrayList;

public class Cliente {
    // Variáveis de instância
    private String nome;
    private int antiguidade;
    private ArrayList<Carrinho> carrinhos;
    private final double VALOR_BASE = 500;

    // Construtor
    public Cliente(String nome, int antiguidade){
        this.nome = nome;
        this.antiguidade = antiguidade;
        carrinhos = new ArrayList<Carrinho>();
    }

    // Métodos seletores
    public String getNome(){
        return nome;
    }
    public int getAntiguidade(){
        return antiguidade;
    }
```

continua

Grupo I - 2


```java
// Método para calcular o crédito
public double calculaCredito(){
    return VALOR_BASE*antiguidade;
}

// Método da relação com a classe Carrinho
public void registaCarrinho(Carrinho carrinho){
    carrinhos.add(carrinho);
}
```

continua

Grupo I - 2


```
    // Método equals. Dois clientes são iguais se tiverem o mesmo nome
    public boolean equals(Object obj) {
        // Verifica se é o próprio
        if (this == obj) return true;
        // Verifica se obj tem referência nula
        if (obj == null) return false;
        // Verifica se os objetos são instâncias da mesma classe
        if (this.getClass() != obj.getClass()) return false;
        // Casting para objeto da classe Produto, pois é Object
        Cliente c = (Cliente) obj;
        // Comparação
        return nome.equals(c.getNome());
    }

    // Método toString
    public String toString(){
        String texto;
        texto = "Nome: "+nome+ " Antiguidade: "+antiguidade+"\n";
        texto += "Número de carrinhos: "+carrinhos.size()+"\n";
        return texto;
    }
}
```

Grupo I - 3

3. Defina a classe **ClientePreferencial**. Redefina o método **calculaCredito**; o crédito de um cliente preferencial é calculado adicionando ao crédito de um cliente normal uma percentagem desse crédito (**percentagem** definida pelo atributo) por cada ano de antiguidade. No entanto, esta adição só é feita se o cliente tiver pelo menos 5 anos de antiguidade; caso contrário, terá o crédito de um cliente normal. Inclua um método **toString** que além do nome, antiguidade e número de carrinhos, imprime também o crédito de um cliente preferencial.

Grupo I - 3


```
public class ClientePreferencial extends Cliente {
    // Variáveis de instância
    private double percentagem;

    // Construtor
    public ClientePreferencial(String nome, int antiguidade, double percentagem){
        super(nome, antiguidade);
        this.percentagem = percentagem;
    }

    // Método que calcula o crédito de um cliente preferencial
    public double calculaCredito(){
        if (getAntiguidade()>= 5)
            return super.calculaCredito()*percentagem*getAntiguidade();
        else
            return super.calculaCredito();
    }

    // Método toString
    public String toString(){
        return super.toString() +"Crédito: " +calculaCredito();
    }
}
```

Grupo I - 4

4. Defina a classe **Carrinho**. **numCarrinhos** guarda o número de carrinhos criados. Inclua o código necessário para atualizar este valor, e um método seletor que devolve este mesmo valor. Tenha em atenção que tem de implementar a relação com a classe **Item** (não é necessário definir a classe Item). Os dois métodos a definir referem-se a esta relação. O primeiro método deve permitir colocar um novo item no carrinho; o segundo deve retirar um item desse carrinho. O método **calculaTotal** calcula o preço total dos items que existem no carrinho.

Grupo I - 4


```
import java.util.ArrayList;

public class Carrinho {
    // Variáveis
    private static int numCarrinhos;
    private ArrayList<Item> items;

    // Construtor
    public Carrinho(){
        numCarrinhos++;
        items = new ArrayList<Item>();
    }

    // Método seletor
    public int getNumCarrinhos(){
        return numCarrinhos;
    }
```

continua

Grupo I - 4


```
    // Método para adicionar items ao Carrinho
    public void adicionaItem(Item item){
        items.add(item);
    }
    // Método para retirar um item do Carrinho
    public void retiraItem(int indice){
        if (indice >= 0 && indice < items.size())
            items.remove(indice);
        else
            System.out.println("Indice incorreto!");
    }

    // Método para calcular o total
    public double calculaTotal(){
        double total = 0;
        for (Item i : items)
            total += i.getPreco();
        return total;
    }
}
```

Grupo I - 5

5. Defina a classe principal. Crie um novo carrinho e coloque dois items nesse carrinho. Imprima a seguinte informação onde X se modifica para cada caso:

O carrinho tem items no valor total de X euros.

Crie um cliente preferencial usando o conceito de polimorfismo; este cliente deverá registar o carrinho anterior. Imprima a informação deste cliente.

Grupo I - 5


```
public class Supermercado {

    public static void main(String[] args) {
        Carrinho carr = new Carrinho();
        Item item;
        item = new Item("1234D", 45.6);
        carr.adicionaItem(item);
        item = new Item("5678P", 5.9);
        carr.adicionaItem(item);

        System.out.println ("O carrinho tem items no valor total de "+carr.calculaTotal()+ " euros.");

        Cliente c = new ClientePreferencial("Ana",3, 25);
        c.registaCarrinho(carr);
        System.out.println(c);
    }
}
```

Grupo II

1. O que são modificadores de visibilidade? Refira quais são utilizados em JAVA e o que cada um destes implica. Dê exemplos de aplicação.

Grupo II

1. O que são modificadores de visibilidade? Refira quais são utilizados em JAVA e o que cada um destes implica. Dê exemplos de aplicação.

2. Descreva o que é o encapsulamento, explique porque é utilizado e como é assegurado.

Grupo II

1. O que são modificadores de visibilidade? Refira quais são utilizados em JAVA e o que cada um destes implica. Dê exemplos de aplicação.

2. Descreva o que é o encapsulamento, explique porque é utilizado e como é assegurado.

3. O que são exceções? Como é que estas podem ser tratadas? E como é que podem ser propagadas?

Grupo II

1. O que são modificadores de visibilidade? Refira quais são utilizados em JAVA e o que cada um destes implica. Dê exemplos de aplicação.

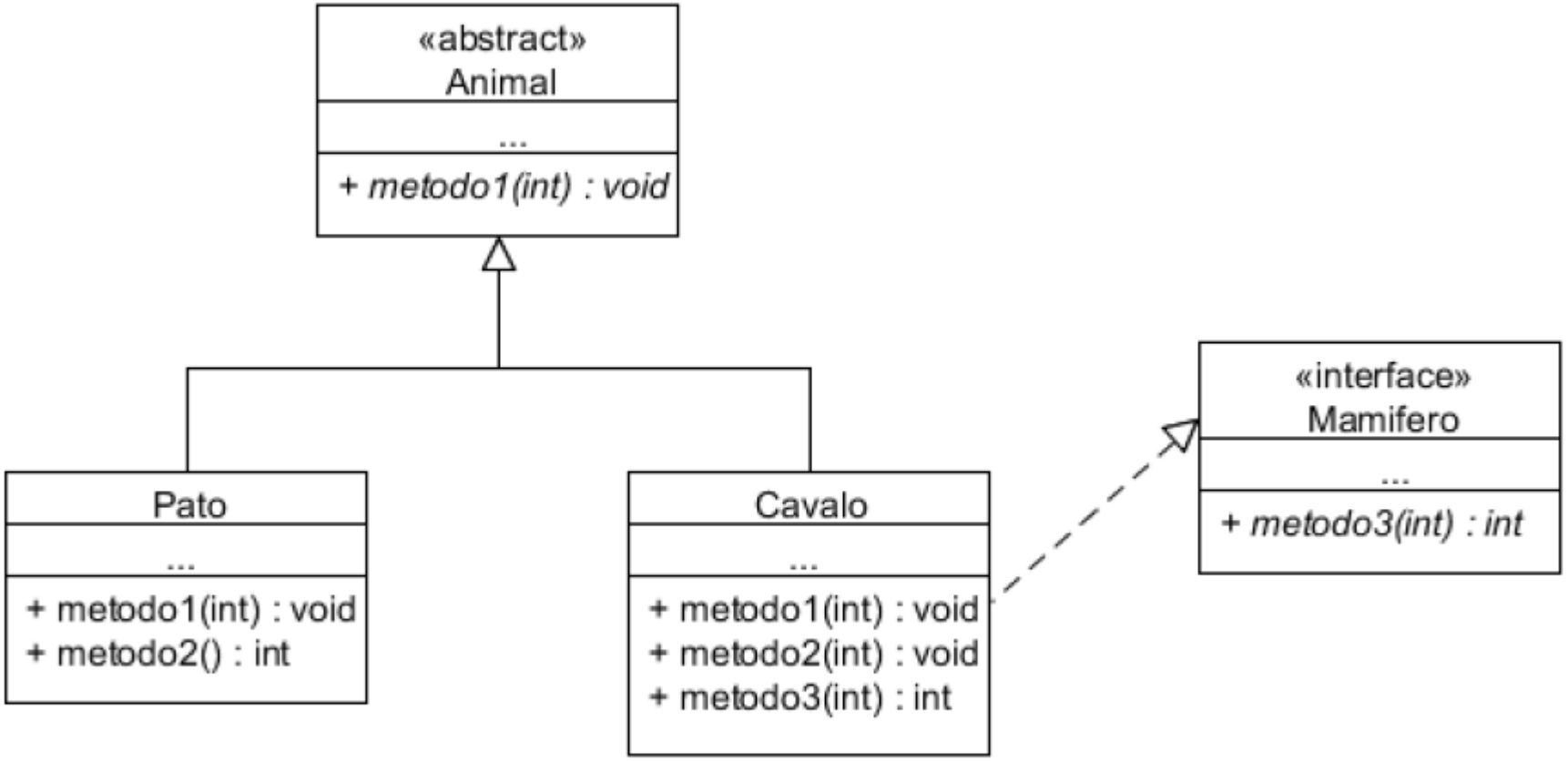
2. Descreva o que é o encapsulamento, explique porque é utilizado e como é assegurado.

3. O que são exceções? Como é que estas podem ser tratadas? E como é que podem ser propagadas?

4. Explique o que é uma classe abstrata, como é definida e qual o seu âmbito de aplicação.

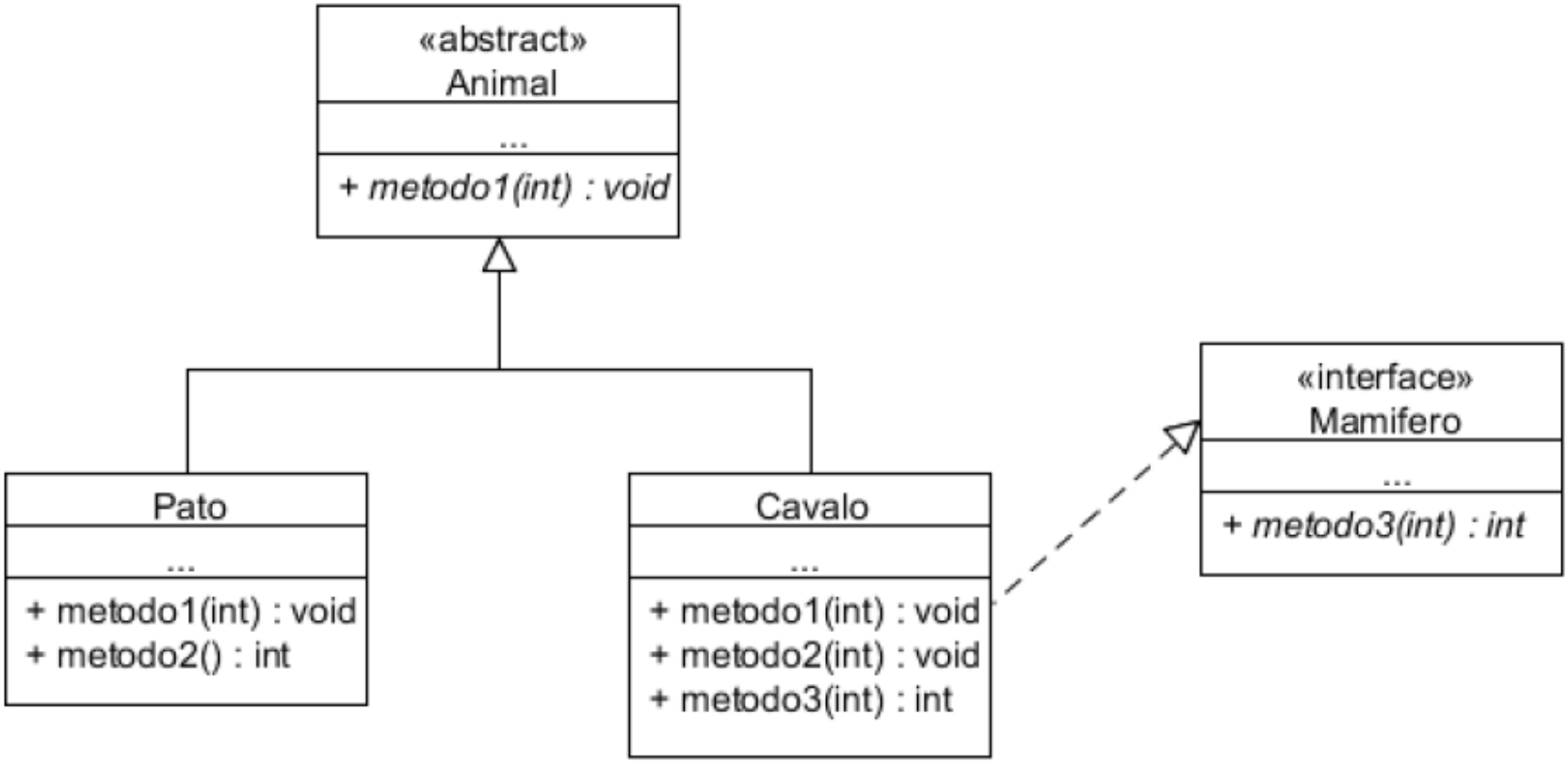
Grupo III

Considere o seguinte diagrama UML:



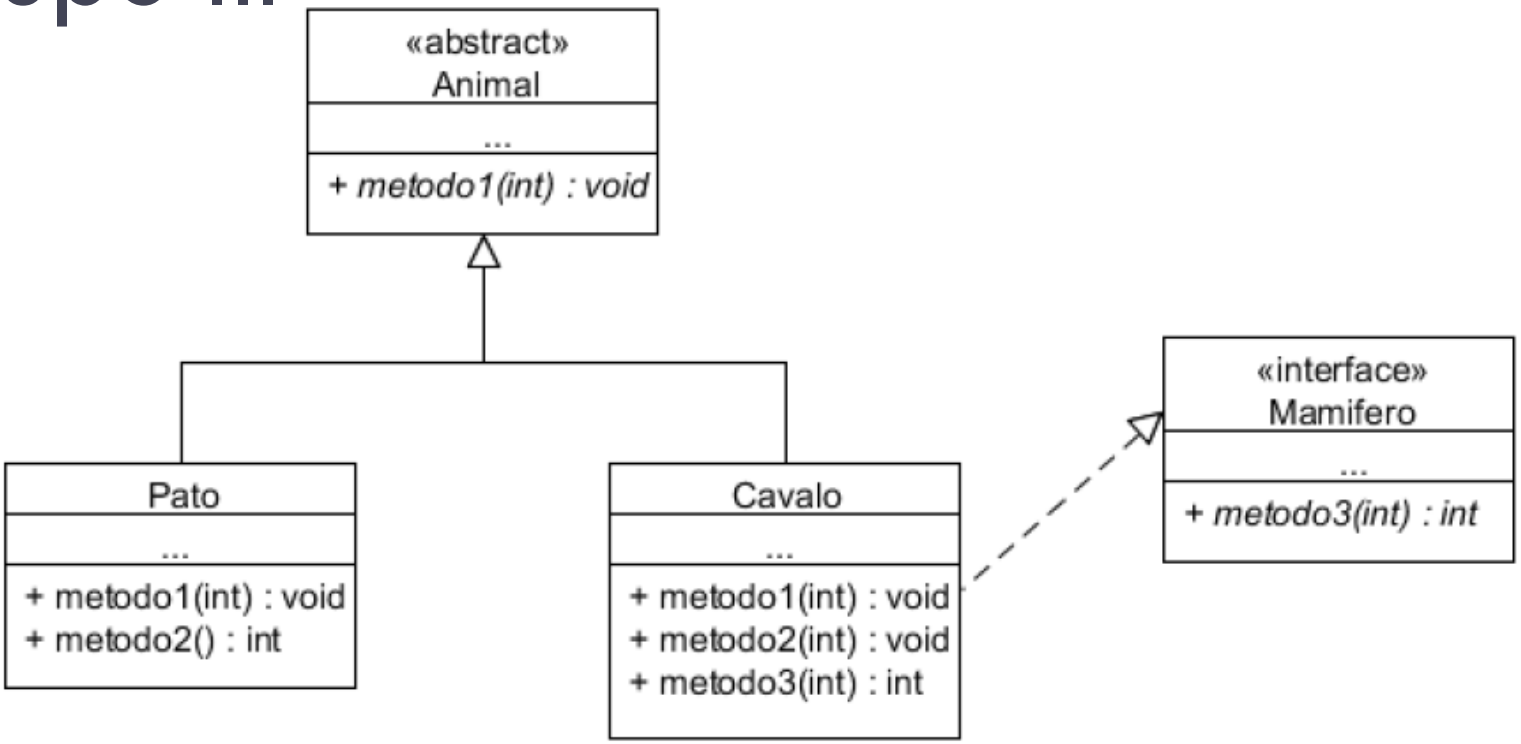
1. Indique o nome e descreva a relação existente entre **Cavalo** e **Mamifero**.

Grupo III



2. Existe neste diagrama algum exemplo de *overloading*? Justifique.

Grupo III

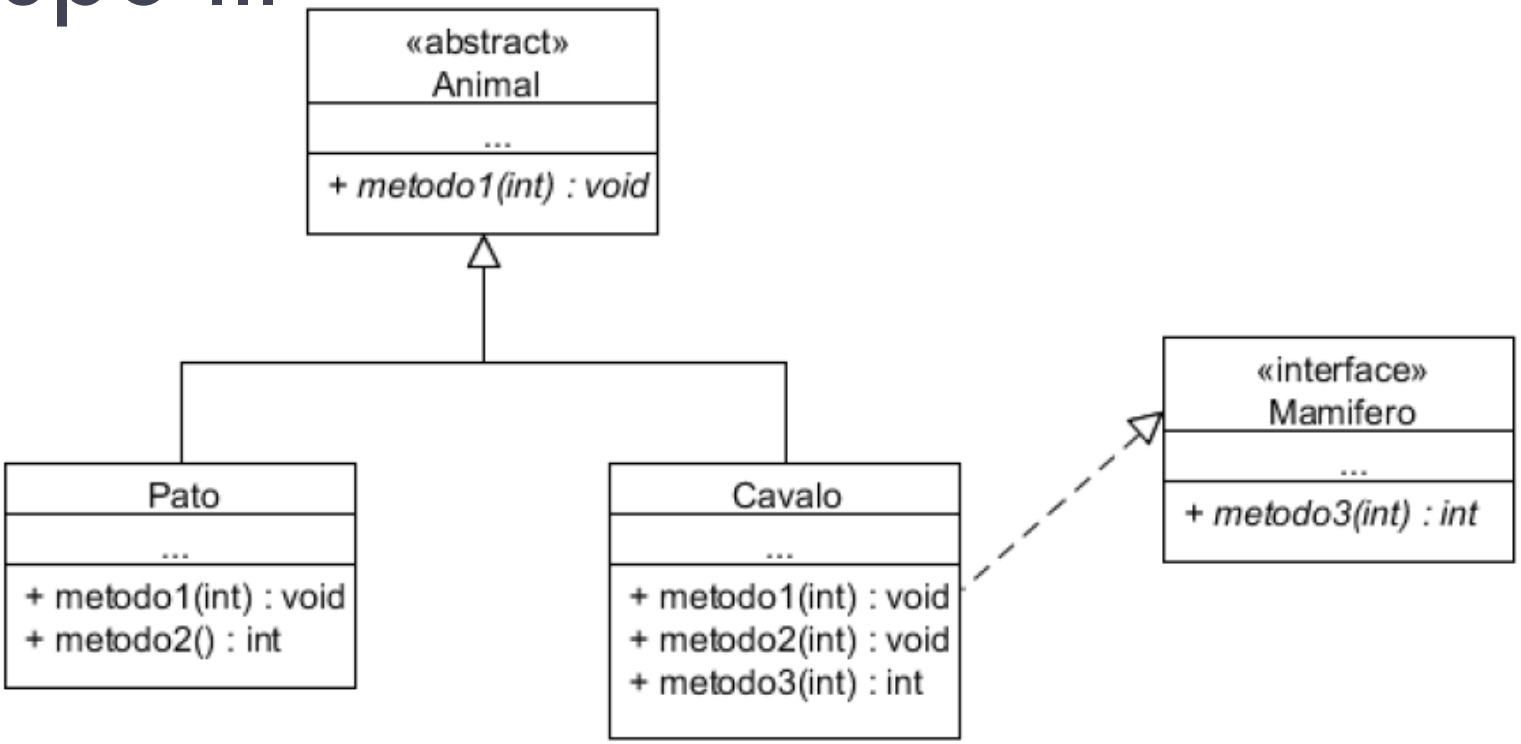


3. Para cada um dos blocos seguintes indique se o mesmo é ou não válido. Deverá justificar **todas** as alíneas.

a.
```
Animal a;
Pato p = new Pato();
a = p;
a.metodo1(5);
```

VÁLIDO. Pelo conceito de polimorfismo, podemos atribuir um subtipo a um supertipo. O metodo1 a ser chamado é o da classe Pato.

Grupo III

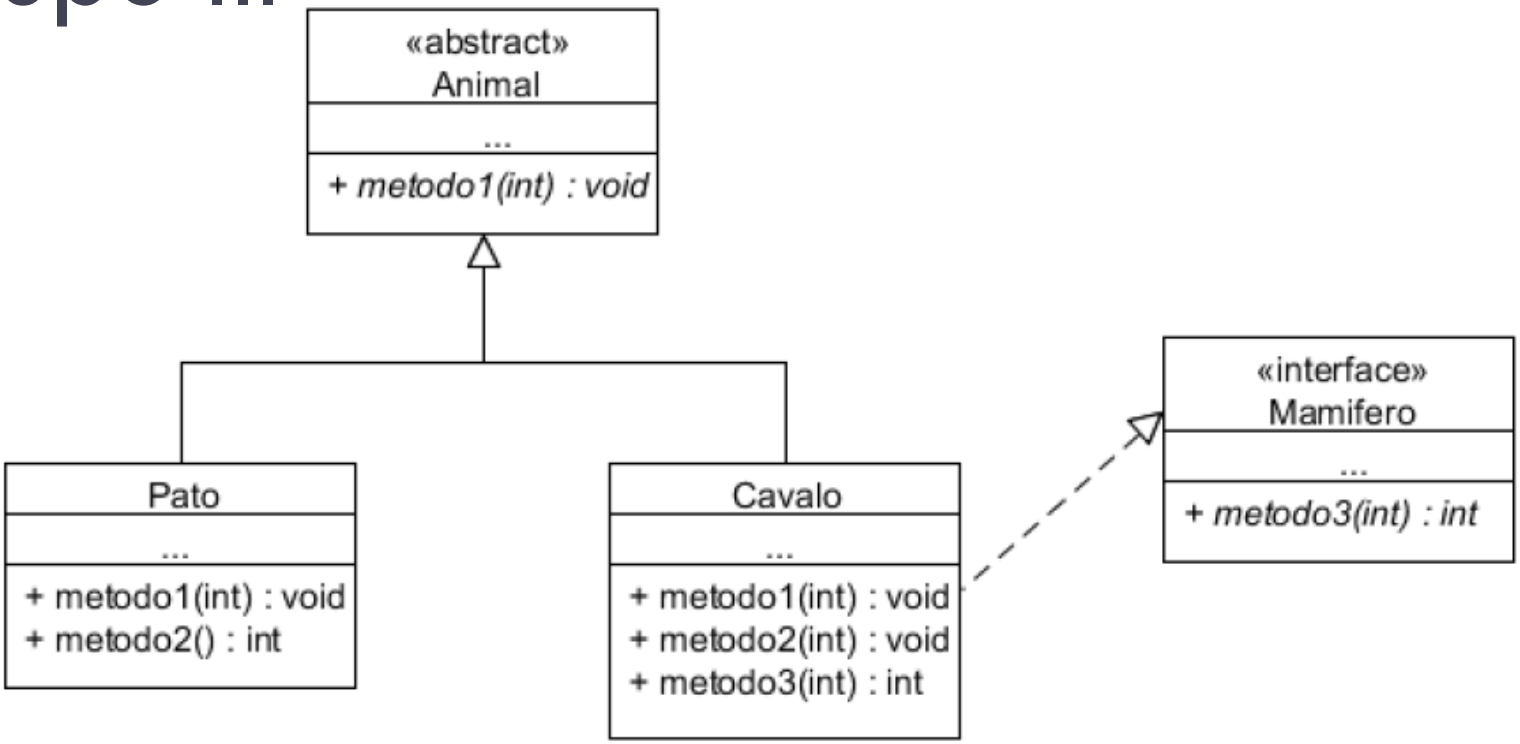


3. Para cada um dos blocos seguintes indique se o mesmo é ou não válido. Deverá justificar **todas** as alíneas.

b.
```
Cavalo c;
Pato p = new Pato();
c = p;
a.metodo1(3);
```

NÃO VÁLIDO. Não podemos atribuir um objeto da classe Pato a uma referência da classe Cavalo, pois são tipos incompatíveis (não há uma relação de herança nem implementação de interface).

Grupo III



3. Para cada um dos blocos seguintes indique se o mesmo é ou não válido. Deverá justificar **todas** as alíneas.

c.
```
Mamifero m;
Mamifero n = new Mamifero();
m = n;
m.metodo3(2);
```

NÃO VÁLIDO. Mamifero é uma interface e como tal não pode ser instanciada.

Grupo IV

1. O código seguinte é susceptível de lançar que exceção(ões)?

```
import java.util.Random;
public class Teste {
   public static void main(String[] args)
{
        Random rand = new Random();
        int [] array =  {1,2,9,5,6,2,3,8};
        int i = rand.nextInt(10);
        int d = rand.nextInt(10);
        System.out.println("Resultado: " +
           (array[i])/d);
   }
  }
```

A - ArrayIndexOutOfBoundsException e ArithmeticException
B - ArrayIndexOutOfBoundsException e NumberFormatException
C - ArithmeticException e NumberFormatException
D - apenas ArithmeticException
E - apenas ArrayIndexOutOfBoundsException

Grupo IV

2. Qual o resultado do código seguinte?

```
public class X {
    public int xpto(){return 5;}
    public int poo() {return 15;}
    public void test(){
        System.out.print(poo()+" ");
        System.out.print(xpto()+" ");
    }
}
public class Y extends X{
    public int xpto(){return 10;}
    public void test(){
        System.out.print(xpto()+" ");
        System.out.print(poo()+" ");
    }
}
public class Teste {
    public static void main(String[] args) {
        X poo = new Y();
        poo.test();
    }
}
```

A - 5 15
B - O código não executa
C - 10 5
D - 10 15
E - Nenhuma das anteriores

Grupo IV

3. Qual o valor de x e de y após a instanciação de dois objetos da classe Z?

```
public class Z {
   private static int x = 0;
   private int y = -1;

   public Z() {
      y = 1;
      x = x + y;
   }
}
```

A - x = -1 e y = -1
B - x = -2 e y = -1
C - x = 2 e y = 1
D - x = 1 e y = 1
E - Nenhuma das anteriores

Grupo IV

4. Que linha(s) dará(ão) um erro de execução?

```
1  public class Teste {
2   public static void main(String[] args) {
3          String a, b, d;
4          int c;
5          a = "12AB345CD";
6          b = a.substring(4,8);
7          c = Integer.parseInt(b);
8          d = a.concat(b);
       }
   }
```

A - 6
B - 7
C - 6, 7
D - 8
E - 7, 8

Grupo IV

5. O HashSet ...

A - é uma coleção organizada e desordenada
B - é uma coleção desorganizada e ordenada
C - é uma coleção organizada e ordenada
D - é uma coleção desorganizada e desordenada
E - não é uma coleção