Universidade de Brasília Faculdade UnB Gama

Orientação a Objetos

Aula 10 - Herança

Daniel Porto

daniel.porto@unb.br

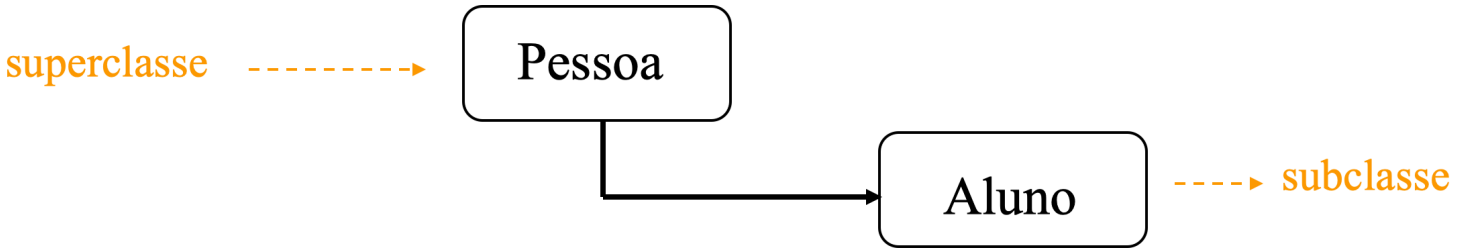
APRESENTAÇÃO

Herança

Sobreposição em Object

HERANÇA

Em POO o termo **herança** faz referência a capacidade de derivação das classes já existentes na geração de novas classes que herdam, direta ou indiretamente, os componentes (atributos e métodos) das primeiras classes.

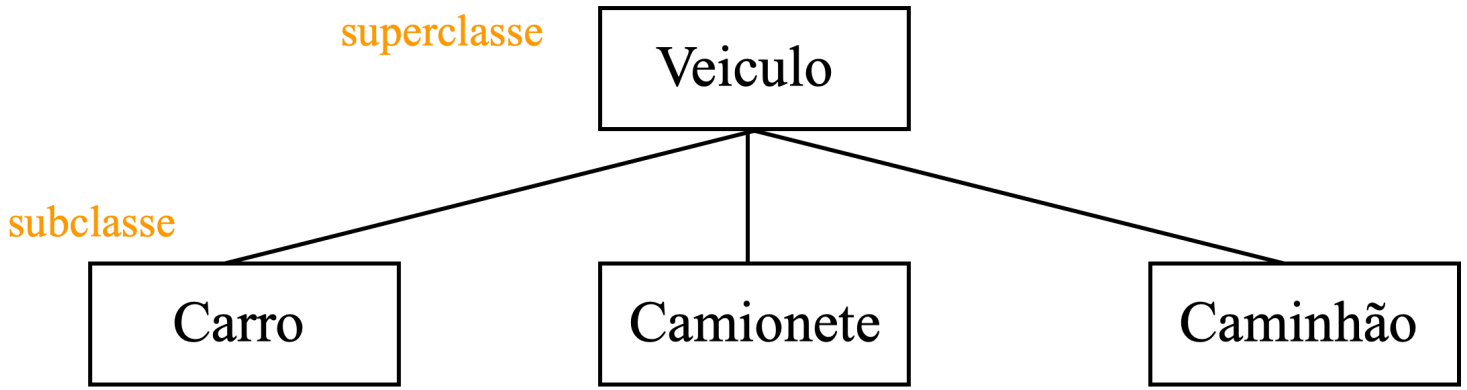


Esta organização na identificação e implementação das classes é extremamente relevante em POO, sendo a superclasse e a subclasse indicada por meio de sinônimos comuns, tais como:

- **superclasse**: base, progenitora, ancestral, pai, ...
- **subclasse**: filho, derivada, subtipo, ...

HERANÇA

Esta organização representa uma hierarquia existente entre as classes, como pode ser observada na representação gráfica:



HERANÇA

Características da Herança

Permite criar novas classes estendendo as classes já existentes.

As subclasses são **especializações** da superclasse e herdam todos os seus componentes (atributos e métodos).

Cada subclasse pode criar novas características (atributos e métodos) além das provenientes de sua superclasse.

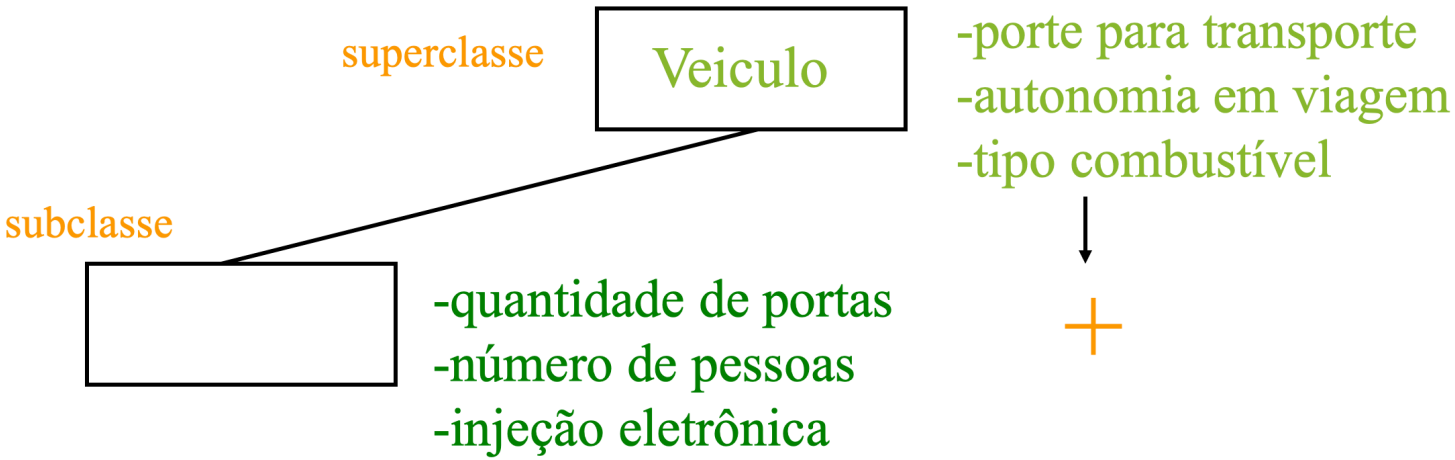
Os métodos da superclasse podem ser reescritos em suas subclasses (sobreposição).

As subclasses podem ser superclasses para outros níveis na hierarquia das classes e na necessidade de manipulação de seus objetos.

Reaproveitamento e reuso de código já criado em outras classes progenitoras ou base.

HERANÇA

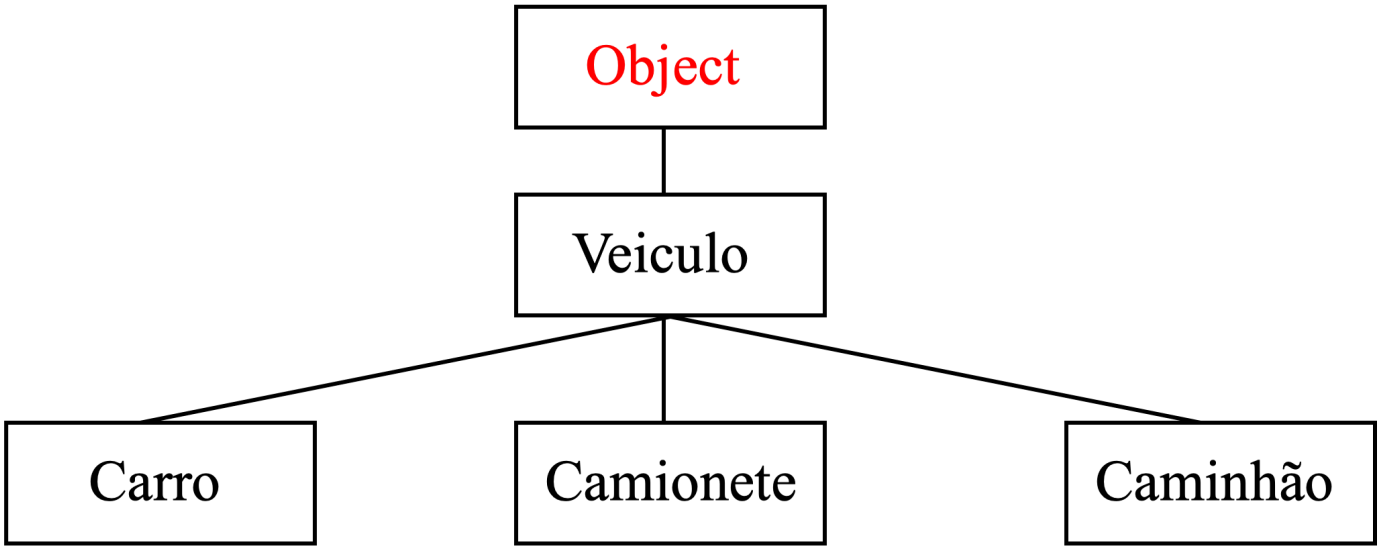
Estas características da herança envolvem a especialização, pois uma nova classe (subclasse) herda as características de uma outra (superclasse), podendo implementar partes específicas que não são contempladas na classe base, tornando-se mais especializada.



Carro (subclasse) consiste na especialização de Veiculo.

HERANÇA

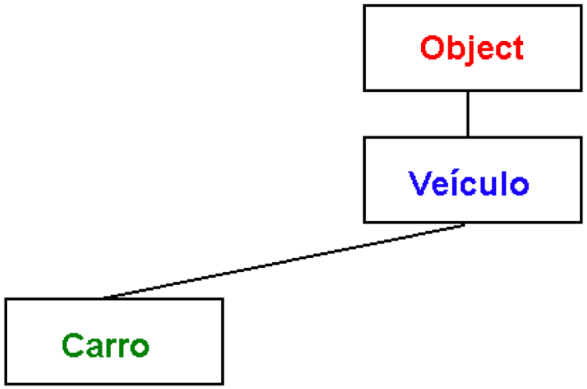
A implementação da herança na POO em Java consiste em uma característica natural, pois toda e qualquer classe elaborada é subclasse da superclasse Object, direta ou indiretamente.



HERANÇA

Observe na representação gráfica da hierarquia de herança anterior que:

- Object é uma classe base ou superclasse
- Veiculo é subclasse **direta** da superclasse Object



- Carro é subclasse **indireta** de Object, mas subclasse **direta** de Veiculo, que por sua vez é superclasse de Carro, apesar de Veiculo ser subclasse **direta** de Object.

HERANÇA

Hierarquia de Herança: Coleção de todas as classes que derivam de uma mesma superclasse comum.

Sequência de Herança: Percurso de uma classe especifica até sua superclasse na hierarquia de herança.

Propriedade da Transitividade:

Veiculo	herda de Object
Carro	herda de Veiculo
↓	↓
Carro	**herda de Object**

HERANÇA

Na linguagem Java a herança direta é realizada por meio da instrução **extends**, não sendo necessária a especificação da mesma para **Object**, que já é padrão.

```
1   /** Síntese
2    *    Métodos: frea(), acelera(), getMarca(),
3    *                getVelocidade(), setMarca(String),
4    *                setVelocidade(float)
5    */
6   public class Veiculo {
7       private String marca;
8       private Float velocidade;
9
10      public void frea() {
11          if (velocidade > 0)
12              velocidade--;
13      }
14
15      public void acelera() {
16          if (velocidade <= 10)
17              velocidade++;
18      }
```

HERANÇA

```
19        // continuação do exemplo anterior
20
21        public String getMarca() {
22            return marca;
23        }
24
25        public void setMarca(String marca) {
26            this.marca = marca;
27        }
28
29        public float getVelocidade() {
30            return velocidade;
31        }
32
33        public void setVelocidade(float velocidade) {
34            this.velocidade = velocidade;
35        }
36    }
```

HERANÇA

```java
/** Síntese
 *   Método: liga(), desliga(), getStatus(),
 *           setStatus(boolean)
 */
public class Carro extends Veiculo {
    private Boolean status;

    public void liga() {
        status = true;
    }

    public void desliga() {
        status = false;
    }

    public boolean getStatus() {
        return status;
    }

    public void setStatus(boolean status) {
        this.status = status;
    }
}
```

HERANÇA

```java
/** Síntese
 *    Objetivo: criar um carro novo no sistema
 *    Entrada:  sem entrada (só atribuições)
 *    Saída:    registro do carro nvo
 */

public class RegistraCarro {
    public static void main(String[] args) {
        Carro auto1 = new Carro();
        // exemplo de inicialização no carro novo
        auto1.setMarca("FIAT");
        auto1.setVelocidade(0);
        auto1.setStatus(false);

        // Mostra carro novo
        System.out.println("Marca= " + auto1.getMarca());
        System.out.print("\tVelocidade= " + auto1.getVelocidade());
        System.out.println("\tSituação= " +
                           (auto1.getStatus() ? "ligado" : "desligado"));
    }
}
```

HERANÇA

Observando o programa anterior tem-se:

- A classe **Veiculo** possui 2 atributos (marca e velocidade) e 2 métodos (frea e acelera)
- Na classe **Carro** existe a necessidade de mais 1 atributo (status – situação de ligado ou não) e 2 métodos (liga e desliga)
- A instrução **extends** estende a classe Veiculo, criando uma nova classe (subclasse) Carro que junta os componentes de Veiculo e as necessidades da Carro
- Ao instanciar o auto1 para Carro (ver no método main) ele possuirá todos os atributos de Veiculo e Carro (3) e todos os seus métodos (4)
- Qualquer alteração na superclasse irá refletir em suas subclasses, por exemplo criando o atributo combustível em Veiculo ela estará disponível para Carro

HERANÇA

Por que SUPERclasse?

O uso da instrução extends estabelece a herança direta em Java, permitindo que subclasses sejam criadas por meio da incorporação de mais componentes (atributos e métodos) do que na classe base, especializando as novas classes derivadas.

Dessa forma, as subclasses possuem mais recursos (componentes) que suas superclasses, não sendo estas últimas (superclasses) tão SUPER assim.

Os prefixos **super** e **sub** são provenientes da linguagem matemática de conjuntos, onde o conjunto de todos os veículos contém o conjunto de todos os carros, sendo Veículo descrito como superconjunto do conjunto Carro, enquanto que Carro é subconjunto do conjunto de todos os veículos.

HERANÇA

Instrução super

Todo construtor de uma subclasse precisa acionar o construtor de sua superclasse, podendo ser este acionamento:

- **explicito**: uso da instrução **super** que aciona o construtor da superclasse
- **implícito**: não aciona, explicitamente, o construtor da superclasse que usará seu construtor padrão definido em Java

A instrução **super** possibilita herança de componentes da superclasse em suas subclasses, sem que tenham que ser redesenvolvidas em cada nova subclasse.

```
public Carro() {
        super(); // aciona construtor da superclasse Veiculo
                :
}
```

HERANÇA

Geralmente, os métodos construtores são sobrecarregados, onde a instrução super pode possuir certa variação:

- **sem parâmetros**: inicia a superclasse com valores padrões
- **com 1 parâmetro String**: título de janela na Swing
- **com parâmetros**: inicia valores de instancia na superclasse

Restrições da Instrução super:

- Deve ser a primeira instrução em um método construtor
- A instrução this e super não podem acontecer, simultaneamente, no mesmo construtor

super - referencia os componentes da superclasse
this - referencia os componentes de sua própria classe

HERANÇA

Observe também, no exemplo anterior, que os métodos definidos em **Veiculo** podem ser usados em **Carro**, sendo automática a disponibilização dos métodos da superclasse para suas subclasses.

Entretanto, pode ser necessária uma **nova definição** para um método herdado da superclasse, mas que tenha um tratamento especializado na subclasse e **oculte** o método da superclasse seguindo a hierarquia das classes.

Para isso é necessário **redefinir o método** no corpo da subclasse (sobreposição). Porém, este método não terá acesso aos atributos privados da superclasse, apesar de cada objeto da subclasse possuir seus próprios atributos herdados da superclasse. Essa alteração só será possível pela subclasse através do uso da interface adequada disponível pela superclasse (garantia de encapsular).

HERANÇA

Supondo a inclusão de um turbo na subclasse Carro, implemente o método acelera() que sobrepõem o método da superclasse Veiculo

```
1   /** Síntese
2    *   Método: liga(), desliga(), acelera(),
3    *           getStatus(), setStatus(boolean)
4    */
5   public class Carro extends Veiculo {
6       private Boolean status;
7
8       public void liga() {
9           status = true;
10      }
11
12      public void desliga() {
13          status = false;
14      }
15
16      public boolean getStatus() {
17          return status;
18      }
```

HERANÇA

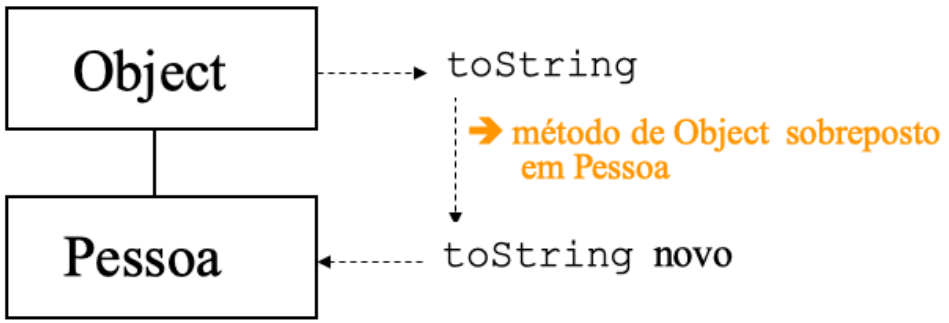
```
19      // continuação do exemplo anterior
20
21      public void setStatus(boolean status) {
22          this.status = status;
23      }
24
25      public void acelera() { // sobrepõem método de Veiculo
26          final int turbo = 2;  // potencia do turbo no carro
27          float novaVelocidade = getVelocidade();
28          if (getVelocidade() <= 10) {
29              novaVelocidade += turbo;
30              setVelocidade(novaVelocidade);
31          }
32      }
33  }
```

Modifique também a classe executável RegistraCarro para acionar o método da subclasse Carro e observe como a aceleração é mais rápida quando usa o método com turbo, **ocultando** o mesmo método de Veiculo.

SOBREPOSIÇÃO DE MÉTODOS DE OBJECT

O recurso da **sobreposição** possibilita a especialização de métodos de qualquer classe que seja superclasse, direta ou indiretamente, de uma subclasse.

Sendo a classe **Object** superclasse de qualquer outra classe (é a classe “suprema”), seus métodos podem ser sobrepostos (redefinidos). Uma situação comum desta sobreposição ocorre com seu método **toString**, definido em **Object** e acionado sempre que uma instrução **print** for executada para mostrar dados de qualquer objeto



SOBREPOSIÇÃO DE MÉTODOS DE OBJECT

toString

Este método é definido na superclasse Object, sendo redefinido (sobreposto) por várias classes para mostrar a situação (valor dos atributos) de seus objetos.

- Pertence a classe suprema em Java - **Object**
- O acionamento da instrução System.out.print o aciona conforme redefinição feita em sua própria classe
- A sobreposição deste método geralmente mostra a situação do objeto que o está acionando, ou seja, faz a representação de todo objeto em uma string a ser mostrada
- Poder ser usado com uma String mutável com todos os valores contidos em seu objeto, inclusive para apresentação de vários objetos de uma mesma classe

SOBREPOSIÇÃO DE MÉTODOS DE OBJECT

```
1   /* Síntese
2    *  Objetivo: cadastrar grupo de pessoas
3    *   Entrada: nome e idade das pessoas
4    *   Saída:   listar todas pessoas cadastradas
5    */
6   import java.util.*;
7   import javax.swing.JOptionPane;
8   public class Principal {
9       public static void main(String[] args) {
10          ArrayList<Pessoa> pessoa = new ArrayList<Pessoa>();
11          String nomeAux;
12          int idadeAux = 0, aux = 0;
13          boolean erro;
14          do {
15              nomeAux = JOptionPane.showInputDialog(null,
16                  "Pessoa "+(aux+1)+"\nDigite seu primeiro nome.",
17                  "Cadastros",JOptionPane.QUESTION_MESSAGE);
18              if(nomeAux.length() != 0) {
19                  do {
20                      erro = false;
21                      try {
22                          idadeAux = Integer.parseInt(
23                          JOptionPane.showInputDialog(null,
24                              "Pessoa "+(aux+1)+"\nDigite sua idade em anos.",
25                              "Cadastros",JOptionPane.QUESTION_MESSAGE));
```

SOBREPOSIÇÃO DE MÉTODOS DE OBJECT

```
26                          if(idadeAux <= 0) {
27                              JOptionPane.showMessageDialog(null,
28                                  "Valor inválido.Informe um número positivo.",
29                                  "Erro",JOptionPane.ERROR_MESSAGE);
30                              erro = true;
31                          }
32                      } catch (NumberFormatException ex) {
33                          erro = true;
34                          JOptionPane.showMessageDialog(null,
35                              "Valor inválido. Informe um número em anos.",
36                              "Erro",JOptionPane.ERROR_MESSAGE);
37                      }
38                  } while (erro);
39                  pessoa.add(new Pessoa(nomeAux,idadeAux));
40                  aux++;
41              }
42          } while(nomeAux.length() != 0);
43          saltaLinha(10);
44          System.out.println("REGISTROS\n\nNOME\tIDADE");
45          System.out.println("================");
46          for(aux = 0; aux < pessoa.size(); aux++)
47              System.out.println(pessoa.get(aux));
48      }
49      public static void saltaLinha(int quantidade) {
50          for(int aux = 0; aux < quantidade; aux++)
51              System.out.println();
52      }
53  }
```

SOBREPOSIÇÃO DE MÉTODOS DE OBJECT

```java
/* Síntese
 *    Atributos: nome, idade
 *    Métodos: getNome(), getIdade(), setNome(String),
 *             setIdade(int), toString(Pessoa)
*/
public class Pessoa  {
    // atributos
    private String nome;
    private Integer idade;
    // método Construtor
    Pessoa(String nomePar, int idadePar) {
        this.setNome(nomePar);
        this.setIdade(idadePar); }
    // métodos para encapsulamento
    public void setNome(String nome) {
        this.nome = nome;    }
    public String getNome() {
        return nome;         }
    public void setIdade(int idade) {
        this.idade = idade; }
    public int getIdade()  {
        return idade;        }
    // método sobrescrito
    public String toString() {
        return(this.getNome() + "\t" + this.getIdade());
    }
}
```

HERANÇA

Restringir a Herança

Além da possibilidade de ocultar um método da superclasse, através da redefinição em sua subclasse, ainda é possível evitar a herança em Classes e Métodos.

O uso da instrução final garante que uma classe não pode ser superclasse ou progenitora de outras classes. Por exemplo: suponha que a classe Camionete seja final.

```
public final class Camionete {
        : // definição dos componentes desta classe
}
```

Todos os métodos de uma classe final, automaticamente, são finais, e não permitem sua redefinição em suas classes derivadas (sobreposição).

HERANÇA

A necessidade de uma lógica em um projeto pode precisar da implementação de somente alguns métodos que não possam ser sobrepostos por suas subclasses. Para isso a palavra reservada final é usada na assinatura do método, evitando que ele seja redefinido. Por exemplo:

```
public final String getCombustivel() {
        : // definição do corpo do método de Veiculo
}
```

Isso oferece segurança quanto a não permitir que um mesmo método, com mesma assinatura, possa ser implementado em uma subclasse com processamento inadequado (classe String em Java é final).

Assinatura de um método não compreende seu retorno!

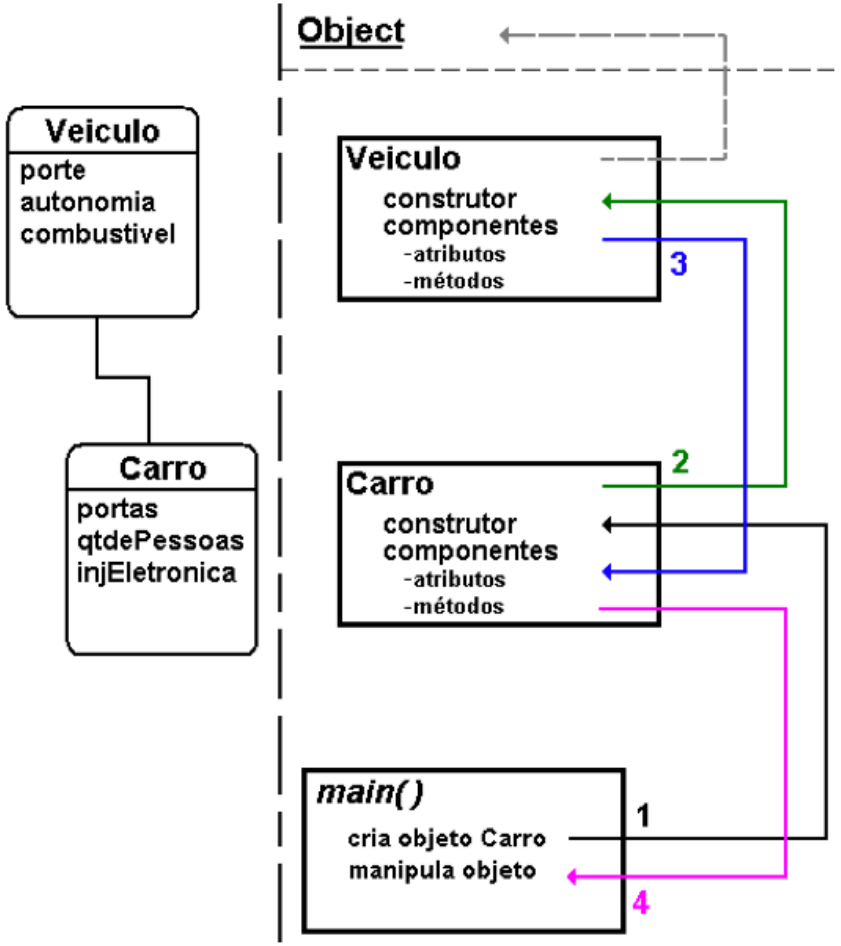
Atenção com os Qualificadores

Um aspecto importante a ser analisado é a definição coerente dos qualificadores de acesso nas classes envolvidas em um projeto e adequadamente definidas em conformidade com estas novas características fundamentais na POO.

Assim como os componentes privados abordados no estudo do encapsulamento, agora também é relevante uma análise precisa com a herança, por exemplo:

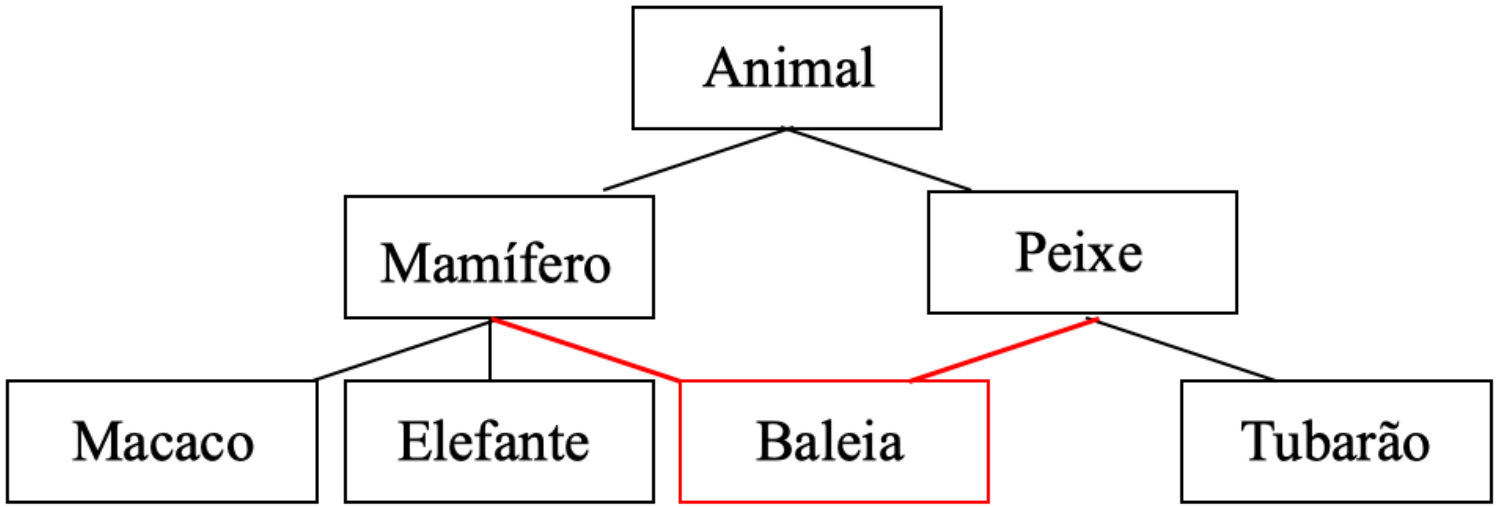
- **private** permite acesso somente da própria classe, seja ela uma superclasse ou subclasse
- **protected** restringe acesso de qualquer classe, mas permite que este acesso aconteça somente pela própria classe e suas subclasses
- **demais qualificadores** permitem acesso das subclasses aos componentes da superclasse, além de outras classes

Representação do Processamento com Herança



HERANÇA

Dentre as várias possibilidades com a herança, a linguagem Java não aceita a implementação da **herança múltipla**, ou seja, uma nova classe derivada ser gerada por mais que uma progenitora (superclasse), por exemplo:



Baleia não pode ser implementada porque possui duas heranças diretas das superclasses Mamífero e Peixe.

HERANÇA

Composição e Herança

A composição em uma classe ocorre quando esta possui como atributo um objeto de outra classe, enquanto que a herança identifica que a classe derivada consiste em uma especialização de sua superclasse. Por exemplo:

A classe Carro possuirá como um de seus atributos um objeto da classe Pneu, sendo geralmente 4 elementos de pneu que são usados por um Carro. Mas é importante observar que **Carro não é Pneu** e por isso não pode estender o mesmo ou vice-versa (Pneu não é Carro e não pode estendê-la).

```
// objeto pneus compõe a classe Carro
public class Carro {
        Pneu [ ] pneus = new Pneu[4];
        :
}
```

```
// classe Carro é derivada de Pneu
public class Carro extends Pneu {
        :
}
```

Situação real incorreta!

HERANÇA

Uma forma de identificar a herança e constatar se ela ocorre realmente é questionar:

A “minha classe” possui esta outra classe, ou ela é esta outra classe?

Outra forma de verificar se a herança é adequada seria averiguar se uma instancia da subclasse pode ser usada como um objeto instanciado da superclasse.

O Carro é um Pneu?
A Camionete é um Veiculo?
Um Carro possui Pneu?
Uma Camionete possui Veiculo?

HERANÇA

É importante destacar que o processo de derivação de uma classe, gerando suas subclasses, pode acontecer até que a abstração necessária para solução de um problema seja atingida. As características pertinentes a estas subclasses não teriam nada a ver com as suas classes irmãs (subclasses em um mesmo nível). Por exemplo:

A geração das subclasses **Carro**, **Camionete** e **Caminhão**, da superclasse **Veiculo**, não possuem características especializadas que as relacione diretamente, não tendo nada a ver uma com a outra.

Apesar desta constatação, é possível que estas subclasses sejam usadas como sua superclasse na:

- atribuição da subclasse em variável da superclasse
- passagem de parâmetro da subclasse para o método que espera a superclasse como parâmetro

HERANÇA

Vantagens com a Herança

Inclusão de novas classes (subclasses) na hierarquia, porém sem qualquer **necessidade de alteração no código**, pois cada classe define suas próprias características.

Reaproveitamento do código criado em uma classe e usado em outra(s) classe(s) derivada(s) (subclasse).

Facilidade na manutenção, que se realizará sobre a classe que possui tal implementação (componentes) e não em vários pontos do código que duplicariam a implementação desta lógica por vários objetos.

Incorporação de características especializadas as necessidades de novas classes e seus objetos (especialização das classes existentes).

Promover a ALTA Coesão e o BAIXO Acoplamento.

Coesão: envolve recursos bem definidos e específicos (bem coeso) a um tratamento computacional.
Acoplamento: relação ou dependência de outra(s) classe(s).

HERANÇA

Propriedades de Acoplamento e de Coesão

O baixo acoplamento, frequentemente, corresponde a um sinal do software elaborado ser bem estruturado e possuir bom design.

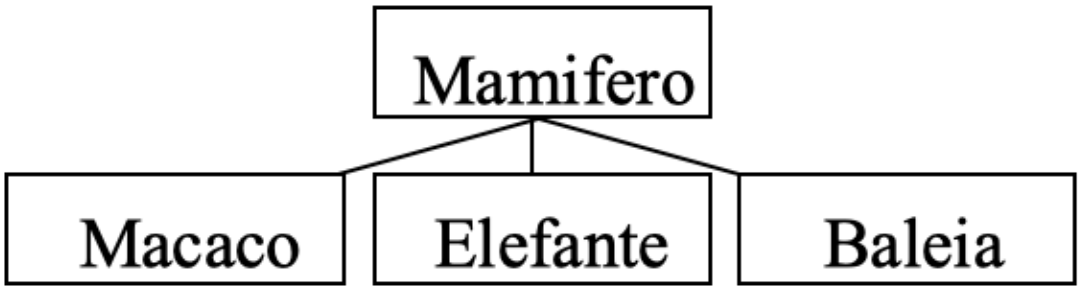
A combinação com a alta coesão fornece suporte aos objetivos gerais de maior legibilidade e facilidade de manutenção no software.

Assim, se é desejado para um software o baixo acoplamento com alta coesão (e vice-versa)

EXERCÍCIOS DE FIXAÇÃO

3) Observando o digrama que representa a hierarquia de heranças abaixo, elabore uma implementação adequada para as classes Mamífero, Macaco, Elefante, Baleia. Implemente na superclasse os componentes (atributos e métodos) necessários para o acompanhamento da idade geral de amamentação materna e de vida do animal (ambos em anos), descrição de sua espécie e o tamanho normal de um na fase adulta (valor em metros), usando ambiente gráfico para leitura destes dados e classes empacotadoras.

Nos macacos indique o porte (pequeno, médio, grande), no elefante e baleia guarde o peso e somente para o elefante também armazene a descrição de seu habitat natural. Implemente os métodos necessários para cadastrar qualquer um destes animais enquanto o usuário quiser e não superar os 500 registros de animais. Quando o usuário não quiser mais fazer registros apresente todos os animais cadastrados como um relatório tabelar na console e encerre o programa.



EXERCÍCIOS DE FIXAÇÃO

4) Uma empresa contrata pessoas registrando seu nome, CPF e data de nascimento, onde todas recebem o mesmo piso salarial de R$232,00. Elabore um programa que permita o cadastramento de várias pessoas como funcionário regular, prestação de serviços ou gerencia de equipe. Implemente para estas 3 possíveis subclasses derivadas de pessoa um método que sobreponha o método da superclasse **calculaSalario** que simplesmente atribui o piso salarial a cada pessoa contratada. Para funcionário regular o método **calculaSalario** deverá fornecer o piso salarial acrescido de 10%, enquanto que a prestação de serviço será calculado o pagamento através da quantidade de horas trabalhadas multiplicada por dois acrescido do próprio piso salarial. Para gerencia de equipe o salário a ser pago será obtido pela quantidade de projetos vezes 50% do piso salarial acrescido do próprio piso. Após o usuário encerrar o cadastro apresente um menu que possibilite ao usuário ter acesso ao total de funcionários cadastrados em cada uma das três categorias e o total salarial a ser pago para mesma. Permaneça neste menu até que o usuário escolha a opção que encerre o programa. Use somente classes empacotadoras nos atributos das classes e colete todos os dados de entrada por janelas gráficas de diálogo.