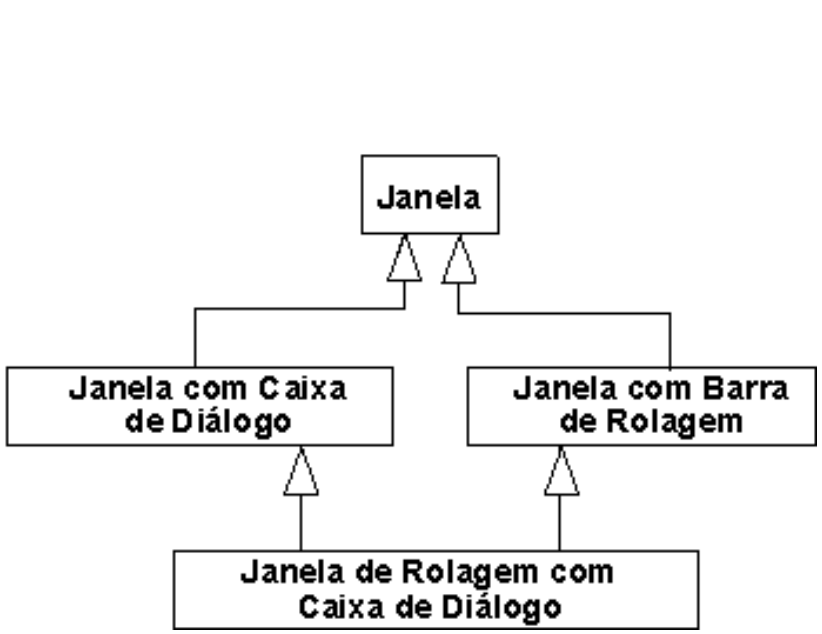
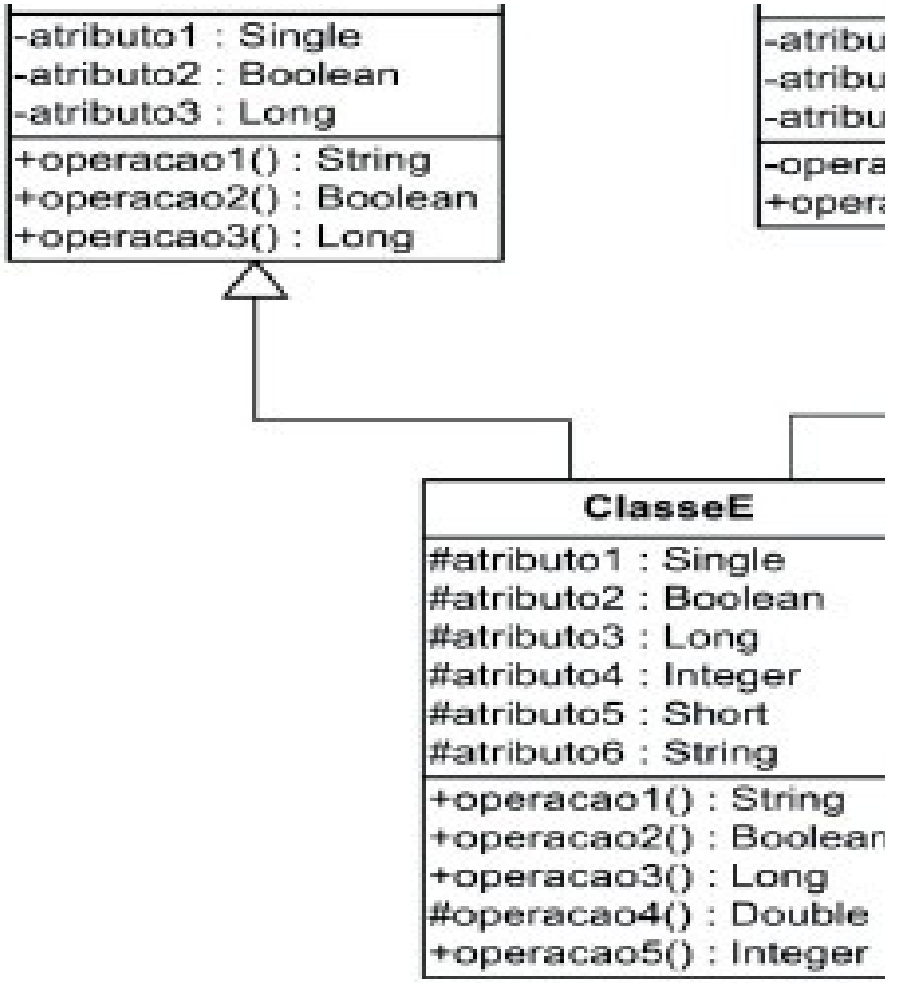
Herança Múltipla, Classes abstratas e Interface

HERANÇA MÚLTIPLA

- Torna possível que uma classe descenda de várias classes





Herança Múltipla, Classes abstratas e Interface

HERANÇA MÚLTIPLA

- Uma classe filha pode ter duas classes pais, herdando as funcionalidades de todas elas

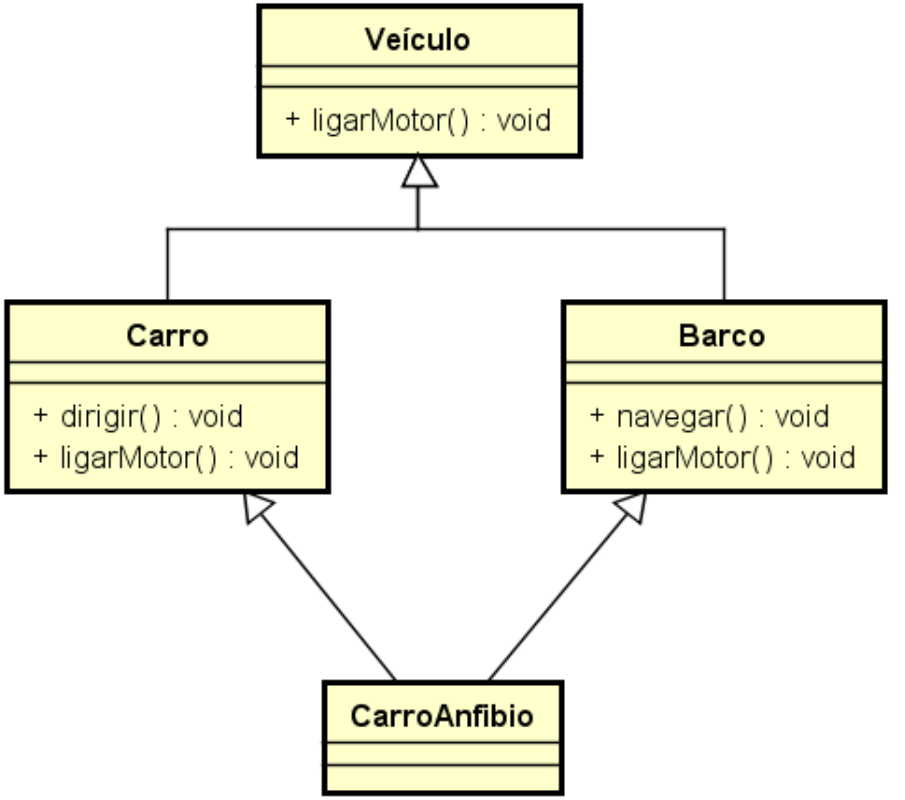
- C++ e Python suportam herança múltipla diretamente, o Java não suporta herança múltipla, mas usa o conceito de interface para simular um comportamento semelhante

- Problemas com Herança Múltiplas
  - Ambiguidade (Problema do diamante)
  - Complexidade (código difícil de entender e manter)

Herança Múltipla, Classes abstratas e Interface

HERANÇA MÚLTIPLA

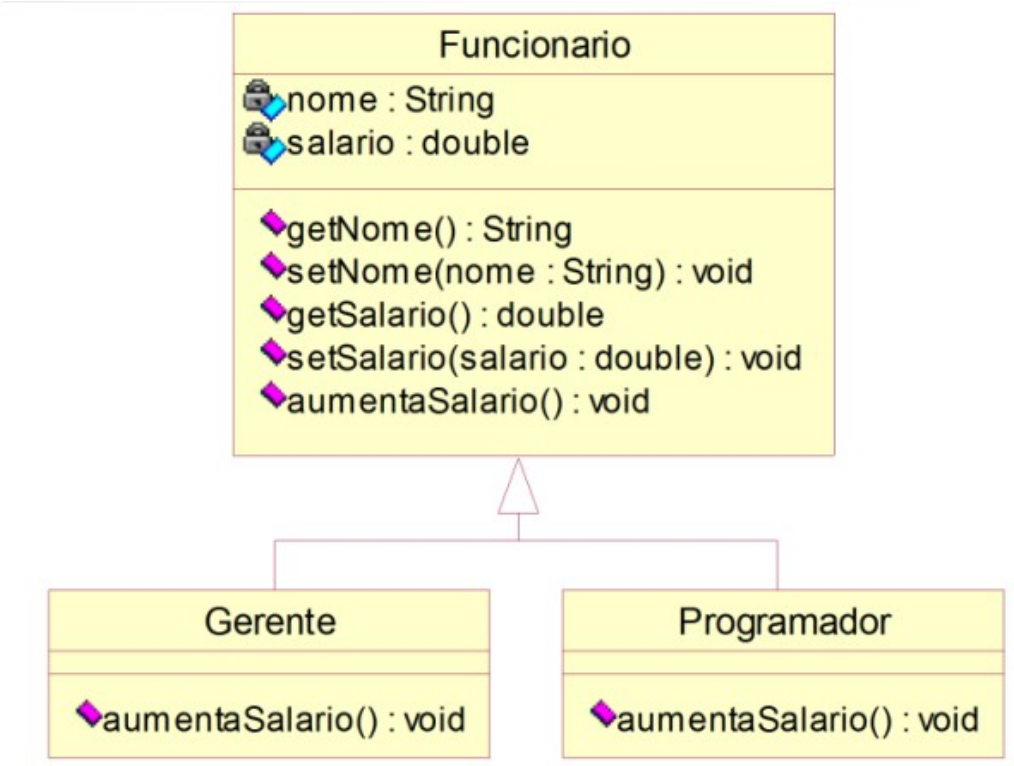
- O problema do diamante
  - Qual método ligarMotor() chamar? (Erro de compilação)
- Complexidade
  - Como chamar o construtor? De carro ou de barco?



Herança Múltipla, Classes abstratas e Interface

Classe Abstrata

- Uma classe abstrata é usada como superclasse para que as subclasses herdem seus atributos e métodos, porém ela não possui objetos, nunca será instanciada
- Uma classe abstrata deve possuir pelo menos um método abstrato (método que não possui corpo, apenas a assinatura e que as filhas devem sobrescrever)



Herança Múltipla, Classes abstratas e Interface

INTERFACE

- É uma coleção de métodos que indica que uma classe possui algum comportamento além do que herda de suas superclasses
- Oferece um conjunto de métodos (sem corpo apenas com assinatura) para que as classes o implementem
- Se uma classe usa uma interface deve, obrigatoriamente implementar todo o seu comportamento (todos os métodos que a interface oferece)

Herança Múltipla e Interface

INTERFACE

- Uma interface é parecida com uma classe
  - porém em uma interface, todos os métodos são públicos e abstratos e se houverem atributos estes são públicos e estáticos
  - A sintaxe para criar uma interface é muito parecida com a sintaxe para criar uma classe: public **interface** <nome_da_interface>{}
  - Vamos fazer uma pausa para olhar o próximo slide...

Pausa para olhar Modificadores de Acesso Java

- *public*: pode ser acessado de qualquer lugar e por qualquer classe

- *private*: não podem ser acessados ou usados por nenhuma outra classe, nem pelas classes herdadas. Um private não se aplica a classes, apenas a métodos e atributos

- *protected*: torna acessível as classes do mesmo pacote ou através de herança (subclasses)

- *default (padrão)*: torna acessível classes do mesmo pacote

- *final*: quando é aplicado na classe não deixa estendê-la, nos métodos impede que o mesmo seja sobrescrito e nos atributos não permite alteração depois de atribuído um valor

Pausa para olhar Modificadores de Acesso Java

- *abstract*: uma classe abstrata não pode ser instanciada e um método abstrato não tem implementação (corpo), só assinatura

- *static*: quando atributos ou métodos são da classe e não dos objetos
  - uma variável estática será criada em todas as instâncias da classe (objetos) e quando seu conteúdo é modificado numa das instâncias a modificação ocorre em todas as demais
  - um método estático pode ser acessado sem a criação de um objeto (não é necessário criar um objeto para acessar o método estático main da classe principal do projeto)

Herança Múltipla, Classes Abstratas e Interface

- **ESTUDO DE CASO**
- Federação Brasileira de Atletismo
  - Nadadores, Corredores, Ciclistas e Triatletas
    - várias características em comum,
      - todos eles devem se aquecer antes da prova.
    - classe que represente todos os tipos de Atletas
    - as demais classes herdarão dessa
    - todo Atleta é uma Pessoa
    - classe para organizar as características comuns a todas as pessoas.

Herança Múltipla, Classes Abstratas e Interface

- **ESTUDO DE CASO**

- Federação Brasileira de Atletismo
  - Classes:
    - Pessoa
      - Atleta
        - Corredor, Nadador, Ciclista e Triatletas

Triatletas não deveriam correr como corredores, nadar como nadadores e pedalar como ciclistas?

Herança Múltipla, Classes Abstratas e Interface

ESTUDO DE CASO

- Federação Brasileira de Atletismo – Hierarquia
  - Todo atleta é uma pessoa. Assim, Atleta herdaria de Pessoa (Em Java: Atleta extends Pessoa)
  - Todo nadador é um atleta. Assim, Nadador herdaria de Atleta (Em Java: Nadador extends Atleta)
  - Todo corredor é um atleta. Assim, Corredor herdaria de Atleta (Em Java: Corredor extends Atleta)
  - Todo ciclista é um atleta. Assim, Ciclista herdaria de Atleta (Em Java: Ciclista extends Atleta)
  - Todo triatleta é nadador, corredor e ciclista. Assim Triatleta deveria herdar de Nadador, Corredor e Atleta.

No caso de Triatleta emprega-se o conceito de **Herança Múltipla**

Herança Múltipla, Classes abstratas e Interface

- Java não implementa herança múltipla por opção
- A herança múltipla pode nos gerar situações inusitadas
- método aquecer() em Atleta
  - método é redefinido em Nadador, Corredor e Ciclista.
  - se esse método não foi implementado em Triatleta
    - Triatleta deveria, por herança, utilizar o método aquecer() de seu ancestral
    - Qual? Triatleta aquece como um corredor, nadador ou ciclista?

- Para evitar esse problema, Java eliminou a possibilidade de herança múltipla fazendo uso de **Interfaces**

Herança Múltipla e Interface

INTERFACE

- Sugestão de resolução do problema:
  - quatro interfaces: Atleta, Corredor, Nadador e Ciclista
  - duas classes: Pessoa (abstrata) e Triatleta.

Herança Múltipla e Interface

Exemplo – Estudo de Caso

- As interfaces Atleta, Nadador, Corredor, Ciclista

```
public interface Atleta {
    public abstract void aquecer();

}
```

```
public interface Nadador extends Atleta {
    public void nadar();

}
```

```
public interface Corredor extends Atleta{
    public void correr();

}
```

```
public interface Ciclista extends Atleta {
    public void pedalar();

}
```

Herança Múltipla e Interface

Exemplo – Estudo de Caso

- A classe Pessoa

```
public abstract class Pessoa {
    private String nome;
    Private String endereço;
    public Pessoa(String nom, String end){
        this.nome = nom;
        this.endereco = end;
    }

    public String getNome(){;
        return this.nome;
    }

    public String getEndereco(){;
        return this.endereco;
    }

  public abstract void imprimirDados();
}
```

Exemplo – Estudo de Caso

- A classe TriAtleta

```
public class TriAtleta extends Pessoa implements Atleta, Ciclista, Corredor, Nadador{

public TriAtleta(String nom, String end){
    super(nom, end);
}
    @Override
    public void pedalar() {
        System.out.println(this.getNome() + " está pedalando");
    }

    @Override
    public void correr() {
      System.out.println(this.getNome() + " está correndo");
    }

    @Override
    public void aquecer() {
      System.out.println(this.getNome() + " está aquecendo");
    }

    @Override
    public void nadar() {
      System.out.println(this.getNome() + " está nadando");
    }

}
```