ORIENTAÇÃO A OBJETOS

AULA 8

Polimorfismo, Classes Abstratas e Interfaces

Vandor Roberto Vilardi Rissoli

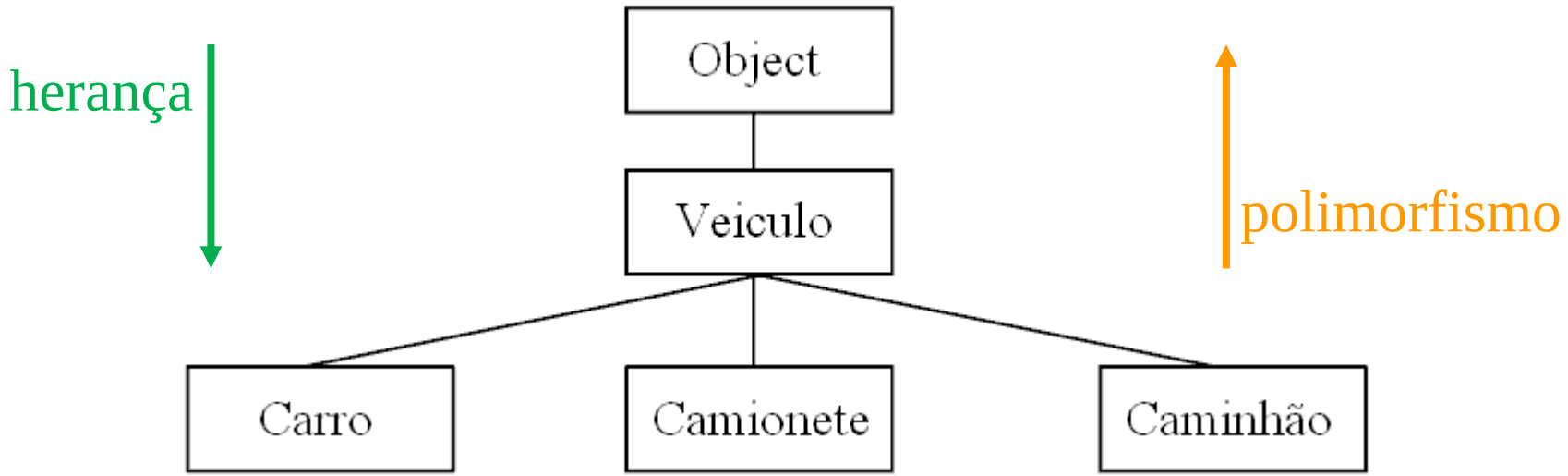
APRESENTAÇÃO

- Polimorfismo
- Classe Abstrata
- Interfaces
- Referências

Polimorfismo

A propriedade de herança fornece um mecanismo de **especialização** interessante a POO, sendo o processo de **generalização** uma outra propriedade importante na POO, denominada **Polimorfismo**.

O polimorfismo trabalha na análise inversa a herança, representada em sua hierarquia de classes, indicando que cada subclasse pode assumir as características e funcionalidades de sua superclasse.



Polimorfismo

Esta propriedade evidencia que uma classe pode assumir diferentes formas (*poli*=várias e *morfo*=fomas), de acordo com suas classes superiores (superclasses).

```
/** Síntese
 *   Conteúdo: estrutura heterogênea de Camionete
 *     - getCarga(), setCarga(boolean), carrega(),
 *     - descarrega()
 */
public class Camionete extends Veiculo {
  private boolean carga;
  public boolean getCarga() {
      return carga;
  }
  public void setCarga(boolean cargaParametro) {
      this.carga = cargaParametro;
  }
  public void carrega() {
      carga = true;
  }
  public void descarrega() {
      carga = false;
  }
}
```

Polimorfismo

A partir da nova classe Camionete, suponha a necessidade de acompanhar a velocidade de um Carro e de uma Camionete em outra classe (UsaVeiculos).

```
/** Síntese
 *    Objetivo: acompanha velocidade de vários veículos
 *    Entrada:  sem entrada (só atribuições)
 *    Saída:    velocidade de cada veículo
 */
public class UsaVeiculos {
  public static void main(String[] args) {
    Carro auto1     = new Carro();      // cria 1 carro
    Camionete auto2 = new Camionete(); // cria 1 camionete
    auto1.setVelocidade(120);
    auto2.setVelocidade(70);
    mostraVelocidade(auto1);  // parâmetro de carro
    mostraVelocidade(auto2);  // parâmetro de camionete
  }
  public static void mostraVelocidade(Veiculo auto) {
    System.out.println("Velocidade = " +
                                    auto.getVelocidade());
  }
}
```

Polimorfismo

Observe que **`UsaVeiculos`** cria 2 objetos diferentes (auto1 é Carro e auto2 é Camionete) e aciona o método mostraVeiculos para mostrar a velocidade destes 2 objetos diferentes.

SOBRECARGA $\neq$ **POLIMORFISMO**

- Parâmetros enviados para mostraVeiculos são de um mesmo tipo (superclasse), não sendo sobrecarga;
- auto1 e auto2 são generalizações de Carro e Camionete, ou seja, os dois são Veiculo;
- O método **mostraVeiculo** pode receber como parâmetro qualquer classe derivada de Veiculo;

$\rightarrow$ Por meio do polimorfismo não foi necessário criar um método para cada tipo de objeto existente nesta classe.

Exercício de Fixação

1) Elabore um programa que possibilite uma interação amigável com seus possíveis usuários onde os mesmos possam cadastrar algumas pessoas com nome e data de nascimento e derive de pessoas as características diferentes para familiar ou amigo. Estas duas especializações serão responsáveis somente pela identificação de grau de parentes para uma pessoa do seu convívio familiar ou para indicação da categoria de relacionamento para um amigo: **0-pessoal**; **1-profissional**; **2-estudantil**.

   Implemente a correta hierarquia entre os elementos envolvidos na solução deste problema, além da classe CadastraPessoa que possibilitará a criação de uma aplicação executável em Java. Permita um cadastro de até 100 pessoas e quando o usuário não quiser mais cadastrar dados apresente as opções de encerrar o programa ou pesquisar um nome desejado para confirmação deste cadastro. Faça um método para esta pesquisa que implemente e use a propriedade do polimorfismo.

Classe Abstrata

A possibilidade de derivação de algumas classes pode exigir que certos comportamentos sejam diferentes em suas subclasses, chegando até a não existirem em algumas. No entanto, os métodos implementados devem possuir uma interface de acionamento padrão (assinatura de métodos) para que tal comportamento seja usado adequadamente por suas classes.

Diante destas situações a linguagem Java permite a implementação de classes abstratas, por meio da instrução ***abstract***.

```
public abstract class Veiculo {
        :
}
```

Classe Abstrata

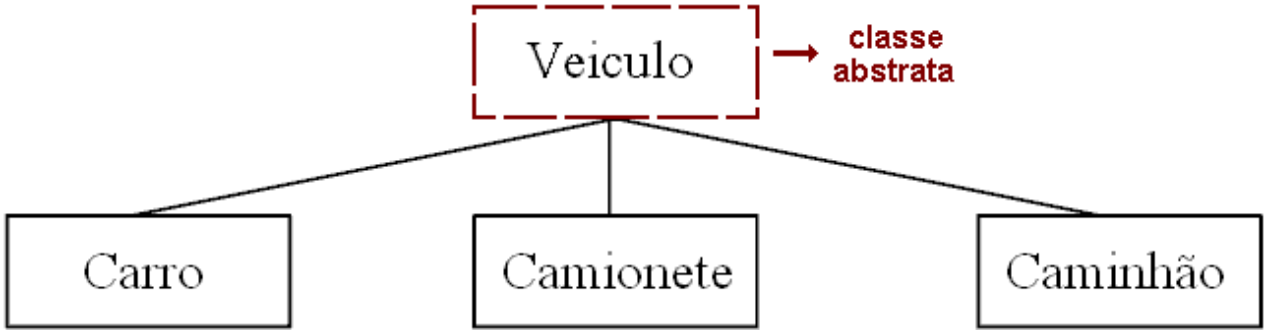
Essa instrução define uma classe, geralmente sinalizando a existência de métodos definidos como abstratos, por meio do uso da instrução ***abstract*** em sua assinatura.

```
public abstract class Veiculo {
    public abstract void mover();
        :
}
```

No entanto, é importante ressaltar que a definição de uma classe abstrata não permite a criação de um objeto a partir dela (classe abstrata), podendo ela possuir **um**, **vários** ou até mesmo **nenhum método abstrato** na definição de seus componentes.

Classe Abstrata

Observe o gráfico que representa a hierarquia de classes que possui Veiculo como uma classe abstrata.



A implementação deste projeto não poderá criar um objeto Veiculo, sendo necessário sua ampliação para criação de uma instancia da mesma.

Observe ainda que podem ser criadas variáveis objeto de uma classe abstrata, tendo essas variáveis que fazer referência a um objeto de uma subclasse não abstrata. Exemplo: `Veiculo auto1 = new Carro();`

→ **auto1** é variável do tipo abstrato **Veiculo** que faz referência a uma instância da subclasse **Carro** que não é abstrata.

Classe Abstrata

```
/** Síntese
 *   Conteúdo: estrutura heterogênea de Veiculo
 *      - frea(),acelera(),getMarca(),getVelocidade()
 *      - setMarca(String),setVelocidade(float)
 *      - abstract mover()
 */
public abstract class Veiculo {
  private String marca;
  private float velocidade;

  // Encapsulando a classe Veiculo
   public String getMarca() {
      return marca;
   }
   public void setMarca(String marca) {
      this.marca = marca;
   }
   public float getVelocidade() {
      return velocidade;
   }
   public void setVelocidade(float velocidade) {
      this.velocidade = velocidade;
   }

   public abstract int mover();   // método abstrato
```

Classe Abstrata

```
// continuação do exemplo anterior
   public void frea() {
     if (velocidade > 0)
       velocidade--;
   }
   public void acelera() {
     if (velocidade <= 250)
       velocidade++;
   }
}
```

//

```
/** Síntese
 *   Conteúdo: estrutura heterogênea de Carro
 *    - liga(), desliga(), getStatus()
 *    - setStatus(boolean), mover()
 */
import javax.swing.JOptionPane;
 public class Carro extends Veiculo {
    private boolean status;

    // Encapsulando a classe Carro
     public boolean getStatus() {
        return status;
     }
```

Classe Abstrata

```
// continuação do exemplo anterior
    public void setStatus(boolean status) {
      this.status = status;
    }
    public void liga() {
      status = true;
    }
    public void desliga() {
      status = false;
    }
    // Implementação obrigatória do método abstrato
    public int mover() {
      final String combustivel = "gasolina";
      return(JOptionPane.showConfirmDialog(null,
         "O veículo está abastecido com " +
         combustivel + "?","Abastecimento",
         JOptionPane.YES_NO_OPTION,
         JOptionPane.QUESTION_MESSAGE));
    }
}
```

//

Classe Abstrata

```
/** Síntese
 *    Objetivo: movimenta um carro
 *    Entrada:  sem entrada (só atribuições)
 *    Saída:    confirma o carro em movimento ou não
 */
public class UsaCarro {
  public static void main(String[] args) {
    Carro auto1     = new Carro();
    auto1.setMarca("FIAT");
    auto1.setVelocidade(0);
    auto1.setStatus(false);

    if(auto1.mover()==0) {  // método abstrato
       auto1.liga();  // método concreto de Veiculo
       auto1.acelera();  // método de Carro
       System.out.println("Carro em movimento");
    }
    else
       System.out.println("Carro precisa de " +
                          "gasolina para se mover.");

    mostraVelocidade(auto1); // parâmetro de carro em
  }                          // método polimórfico
```

Classe Abstrata

```
// continuação do exemplo anterior

  public static void mostraVelocidade(Veiculo auto)
  {
     System.out.println("Velocidade = " +
         auto.getVelocidade()); // método de Veiculo
  }
}
```

- A classe **Veiculo** possui um método abstrato (mover), precisando ser definida como uma classe abstrata
- O método mover() é definido como abstrato em **Veiculo**, onde somente possuirá a assinatura do método com a instrução **abstract**
- Todas as classes não abstratas, derivadas de **Veiculo**, são obrigadas a implementarem o método mover()
- A classe abstrata **Veiculo** possui componentes concretos, mas não pode gerar seus próprios objetos
- **Carro** é derivada de **Veiculo** e tem que implementar o método mover(), definido como abstrato em **Veiculo**

Classe Abstrata

O uso de classes abstratas possibilita a declaração de classes que **definam apenas parte de uma implementação**, deixando para suas subclasses efetuarem a implementação adequada e específica de alguns métodos necessários para algumas especializações, podendo todos os seus métodos também serem abstratos.

A definição de qualquer classe pode sobrepor métodos de sua superclasse para declará-los como abstratos, **tornando um método concreto em abstrato** naquele ponto da hierarquia das classes. Esta possibilidade é interessante para as situações que envolvem uma implementação padrão de classe inválida para uma parte da hierarquia de classes.

Exercício de Fixação

2) Implemente a classe Veiculo como uma classe abstrata que contem seus métodos abstratos acelera e mover respeitando as seguintes características para as 3 subclasses Carro, Camionete, Caminhão, além da **TestaVeiculo** que conterá os recursos para criação de uma aplicação executável Java.

a) todo o carro é movido a gasolina e se tiver abastecido deverá ser solicitado ao usuário qual a potência do motor (1.0, 1.4, 1.6, 1.8 ou 2.0). Após estas informações o carro deverá ser ligado e colocado em movimento com a multiplicação de sua potência por 1 (carro se coloca em movimento). Caso o mesmo não tenha sido abastecido não deve ser solicitada a potencia porque o carro permanecerá sem movimento (velocidade zero que será apresentada ao usuário);

Exercício de Fixação

Continuação do exercício 1

b) para camionete deve ser solicitado ao usuário se a potência da mesma é 2.2, 2.8 ou 4.3, sendo ligada após esta mensagem e colocada em movimento com a multiplicação de potência por 1 (camionete se coloca em movimento);

c) no caminhão deve ser solicitado ao usuário se o mesmo esta carregado ou não, onde se ele estiver deverá ser ligado e colocado em movimento com velocidade 1 e senão estiver carregado ele também deverá ser ligado, mas colocado em movimento com velocidade 1.5.

Elabore um programa que verifique qual veículo o usuário gostaria de testar e solicite as informações necessárias para o veículo informado, ligando e o colocando em movimento se for possível pela situação momentânea do veículo escolhido.

Interfaces

Um tipo de recurso especial que só possui **métodos abstratos** e **atributos finais** ou **static** (constantes), são conhecidas como Interfaces em Java. Por meio destas é definido um padrão a ser implementado, além do caminho público para especificação do comportamento de classes (métodos).

Independente de suas posições hierárquicas, as interfaces implementam **comportamentos comuns** entre diferentes classes.

Cada classe pode implementar uma ou mais Interfaces, definindo implementações específicas para cada um de seus métodos abstratos que possuam assinaturas definidas dentro de suas respectivas Interfaces.

Interfaces

Uma interface é usada quando existe a necessidade de implementação de métodos similares em classes não relacionadas. Exemplo: tratar o peso de Carro e Elefante.

Outra razão consiste no mecanismo existente em Java para trabalhar a **herança múltipla**, essencial na solução de alguns tipos de problemas computacionais.

Esta programação, usando interface, é realizada com emprego das instruções: **interface** (cria uma Interface) e **implements** (implementa em uma classe uma ou várias interfaces), por exemplo:

```
interface Tamanhos {
                    :
                   }
public class Animal implements Tamanhos {
    :
}
```

Interfaces

Características das Interfaces

Todas as declarações internas de uma interface são públicas, podendo serem usadas no pacote onde foram declaradas. Caso a interface também seja definida como pública, qualquer outro pacote poderá implementá-la, fazendo uso de seus componentes.

Atributos nas Interfaces

- Atributos nas interfaces são definidos como constantes
- Implicitamente os atributos são sempre ***final*** e ***static***, não sendo necessário o uso destas instruções em sua definição

Interfaces

Métodos nas Interfaces

- Métodos em uma interface são sempre públicos
- Todos são abstratos e possuem somente a definição de suas assinaturas
- ***public*** e ***abstract***, geralmente não são usados na declaração de métodos em interface, pois já são implícitos
- Métodos de interface nunca podem ser ***static***, pois indicam que são específicos de uma classe e nunca serão ***abstract***
- Cada classe que implementa a interface deve implementar todos os seus métodos, definindo suas ações (seu corpo)
- As classes que implementam somente alguns dos métodos de uma interface devem ser declaradas como abstratas

Interfaces

Classes Abstratas e Interfaces

A definição de uma **classe abstrata** pode conter métodos concretos (implementados) e abstratos (só assinaturas a serem implementadas em suas subclasses), enquanto que na **interface** todos os métodos só podem ser abstratos (só possui assinaturas a serem implementadas em suas subclasses).

Apesar de algumas diferenças, as interfaces possuem em comum, com as classes, 2 características importantes:

- ambas definem **tipos** a serem usados na programação
- classes e interfaces também podem declarar **novos métodos**, porém as interfaces não podem implementá-los, pois são abstratos (só assinaturas)

Interfaces

A declaração de uma interface inclui um novo tipo, mas nele não existem valores, pois nas interfaces não são definidas, realmente, a implementação de nenhum método, sendo especificado apenas suas assinaturas.

A criação de objetos de classes, que implementem a interface, devem definir as ações de processamento que cada método realizará em seu acionamento.

A seguir, observe o exemplo que esta utilizando a classe abstrata Veiculo, bem encapsulada no exemplo de código anterior. No entanto, a subclasse Carro esta sendo elaborada novamente por receber modificações relevantes, incluindo a inclusão de uma interface.

Interfaces

// Inclua neste projeto a classe abstrata Veiculo

```
/** Síntese da Interface
 *     Atributo: limite
 *     Método: controlaVelocidade()
 */

public interface Rodovia {
  final float LIMITE = 120;    // atributo constante
  // todo método é abstrato na interface (assinatura)
  public boolean controlaVelocidade(float velocidade);
```

//

```
/**Síntese
 *   Atributos:
 *   Métodos: liga(), desliga(), getStatus() , mover()
 *         setStatus(boolean),controlaVelocidade(float)
 */
import javax.swing.JOptionPane;
```

Interfaces

```
// continuação do exemplo anterior
public class Carro extends Veiculo implements Rodovia {
  private boolean status;
  public boolean getStatus() {
    return status;
  }
  public void setStatus(boolean status) {
    this.status = status;
  }
  public void liga() {
    status = true;
  }
  public void desliga() {
    status = false;
  }
  public int mover() {
    final String combustivel = "gasolina";
    return(JOptionPane.showConfirmDialog(null,
           "O veículo está abastecido com " +
            combustivel + "?","Abastecimento",
           JOptionPane.YES_NO_OPTION,
           JOptionPane.QUESTION_MESSAGE));
  }
```

Interfaces

```
// continuação do exemplo anterior
 // Implementação obrigatória do método da interface
  public boolean controlaVelocidade(float velocidade) {
    if(velocidade < LIMITE)
       return true;
    else
       return false;
  }
}
```

//

```
/** Síntese
 *    Objetivo: movimenta um carro
 *    Entrada:  situação da aceleração e combustível
 *    Saída:    mostra movimentação do carro
 */
import javax.swing.JOptionPane;
public class UsaNovoCarro {
  public static void main(String[] args) {
    Carro auto1 = new Carro();
    auto1.setMarca("FORD");
    auto1.setVelocidade(0);
    auto1.setStatus(false);
```

Interfaces

```
// continuação do exemplo anterior
   if(auto1.mover()==0) {
     int continua;
     auto1.liga();
     auto1.acelera();
     System.out.println("Carro em movimento");
     mostraVelocidade(auto1);
     do {
        for(int aux=0; aux<10; aux++) {
          auto1.acelera();
          mostraVelocidade(auto1);
        }
        System.out.println();
        continua = JOptionPane.showConfirmDialog(null,
                          "Deseja continuar acelerando?",
                "Aceleração", JOptionPane.YES_NO_OPTION,
                              JOptionPane.QUESTION_MESSAGE);
     } while (continua == 0 &&
      auto1.controlaVelocidade(auto1.getVelocidade()));
   }
   else {
```

Interfaces

```
// continuação do exemplo anterior
        System.out.print("Velocidade = ");
        mostraVelocidade(auto1);
        JOptionPane.showMessageDialog(null, "Carro " +
                    " precisa de gasolina para se  mover.",
                "Informação",JOptionPane.WARNING_MESSAGE);
      }
    }

    public static void mostraVelocidade(Veiculo auto) {
      System.out.print("\t" + auto.getVelocidade());
    }
}
```

Interfaces

Herança em Interfaces

Similar as classes, que podem herdar métodos e atributos, as interfaces também podem herdar assinaturas de métodos e constantes de outras interfaces, sendo esta herança indicada pela instrução ***extends*** entre interfaces.

Apesar das Interfaces não serem partes de uma hierarquia de classes, elas podem:

- relacionar-se entre si (estendendo uma interface - superinterface e subinterface) através da ***extends***
- permitir que classes não relacionadas (sem herança direta) implementem a mesma interface

Interfaces

Suponha a extensão da interface Rodovia para uma subinterface chamada Rua, onde o limite permitido é 50% da velocidade máxima definida na Rodovia.

Esta nova interface possuiria um novo método (só assinatura) que obrigaria suas classes implementadoras a averiguarem o local de movimento do Carro em questão (rodovia ou rua) para acompanhar seu limite.

```
public interface Rua extends Rodovia {
   public float ajustaLimite(float maxVelocidade);
}
```

→ O uso da nova interface exige alterações no projeto, mas permite que qualquer classe não relacionada com Carro ou mesmo Veiculo possam utilizá-las para análise de velocidade, mesmo não sendo relacionadas com este projeto.

Interfaces

Novos Tipos

As classes derivadas e as interfaces implementadas em uma classe progenitora são coletivamente denominadas supertipos, enquanto as novas classes estendidas deste supertipo são um subtipo.

O tipo completo desta nova classe inclui todos os seus supertipos, de forma que uma referência a um de seus objetos pode ser utilizado polimorficamente, com referência a qualquer um de seus supertipos (classes ou interfaces).

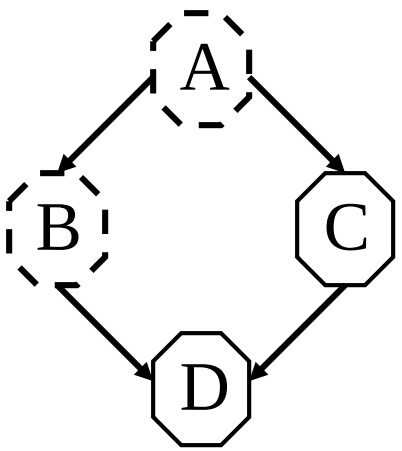
As definições de interface criam novos tipos, como ocorre com as classes, podendo estes tipos serem usados para definir variáveis, que recebem objetos que implementem estas interfaces (polimorfismo).

Interfaces

Herança Múltipla em Java

Java disponibiliza a herança simples, que impede a implementação de alguns projetos que necessitam das características vinculadas a herança múltipla.

No entanto, Java incorpora a herança múltipla de forma alternativa, com o uso de interface, evitando um dos principais problemas de implementação deste tipo de herança, que pode ser representado no “losango da herança”.

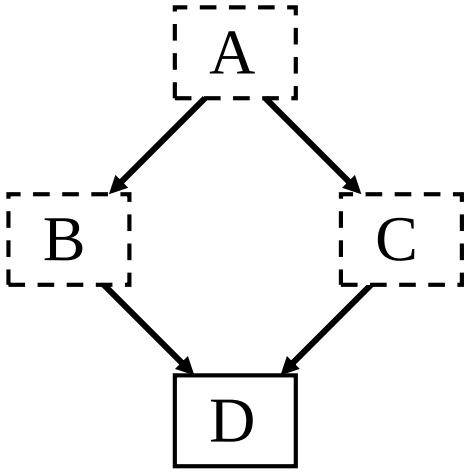


Suponha **A** e **B** interfaces que não possuem implementação e não geram conflitos sobre os componentes herdados para um objeto de **D** que ainda herda da classe **C**.

Interfaces

Uma outra situação relacionada a este losango poderia ser a criação de uma nova classe D proveniente somente das interfaces A, B e C (interfaces gerando nova classe).

```
interface A {
   : // codificação adequada
}
interface B extends A {
  : // codificação adequada
}
interface C extends A {
  : // codificação adequada
}
class D implements B, C {
    :  // codificação da classe
}
```



Assim, seria criada uma única classe (D) pertencente a hierarquia das classes, sendo ela baseada somente em interfaces.

Interfaces

Diferentemente das classes, uma interface **não** possui uma **raiz**, como ***Object*** para classes, mas qualquer tipo de interface pode ser passada como parâmetro para um método que tenha um argumento do tipo ***Object***.

Isso é possível porque a implementação da interface exigi a criação de um objeto e este deve ser de alguma classe para ser implementado.

Sendo todas as classes subclasses de ***Object***, esta operação é válida e funcionará corretamente pelo polimorfismo.

Interfaces

Uma implementação comum em Java, que possibilita a herança múltipla, consiste em classes derivadas que implementam uma ou mais interfaces, por exemplo:

```
public class Aluno extends Pessoa implements Frequencia, Avaliacao
{
        :                    // componentes e programação da classe
}
```

- Aluno é subclasse direta de Pessoa e recebe sua herança
- Aluno implementa as interfaces Frequencia e Avaliacao em sua codificação (instruções dos métodos abstratos nas interfaces sendo definidos na própria classe Aluno)
- Apesar de Frenquencia e Avaliacao não pertencerem a hierarquia de classes, Pessoa e sua nova subclasse Aluno pertencem

Interfaces

Quando Usar Interface

- Definição de padrões de métodos para diferentes classes
- Implementação das características de herança múltipla
- A necessidade de extensão de uma classe, abstrata ou não, deve ser implementada como interface facilitando possíveis extensões

As diferenças entre a implementação de classes ou interfaces, normalmente direcionam a escolha adequada de qual o melhor recurso a ser usado em uma implementação específica.

A **classe abstrata** permite o fornecimento de algumas ou todas implementações, facilitando sua herança, enquanto que as **interfaces** exigem seu repasse, por vezes tedioso e propenso a erros de implementação.

Exercícios de Fixação

3) Implemente interfaces que sejam herdadas da superinterface Rodovia, apresentada nos exemplos desta aula. Depois evolua o exemplo da aula para permitir que um Carro qualquer, com dados informados pelo usuário, possa transitar por uma rodovia com limite já definido; por uma rua com 50% da velocidade limite da rodovia e por uma avenida com limite máximo de 80 km/h. Permita que o usuário cadastre um carro especifico e acelere o quanto quiser, podendo o mesmo abastecer o carro quando este estiver sem gasolina.
   Faça uso de todos os conceitos de POO estudados até o momento e use a interface gráfica para interagir com o usuário como o exemplo de código anterior já vem fazendo.

Exercícios de Fixação

4) Faça um programa que armazene dados de Empregados cadastrando sua matrícula funcional, nome completo e salário. Implemente também um controle para Terrenos na região, onde deverá ser armazenado os dados do endereço em uma única StringBuilder com o endereço completo, a área ocupada em um valor inteiro de metros quadrados e valor atual em reais (R$). Elabore uma interface, denominada **Analise**, que poderá verificar o menor e maior valor do tipo real, a existencia ou não de duplicidade entre valores inteiros, o somatório de qualquer tipo de dado numérico e a média de quaisquer valores numéricos.

   Forneça opções de menu para o usuário aplicar esta interface e fazer análises correspondentes a qualquer empregado ou terreno, conforme ele deseje e faça tal análises somente por meio da interface **Analise**. Se o usuário quiser encerrar o programa sua solução deverá mostrar todos os dados cadastrados na console como um relatório tabelar.

Referência de Criação e Apoio ao Estudo

Material para Consulta e Apoio ao Conteúdo

- HORSTMANN, C. S., CORNELL, G., Core Java2 , volume 1, Makron Books, 2001.
  - Capítulo 5 e 6
- FURGERI, S., Java 2: Ensino Didático: Desenvolvendo e Implementando Aplicações, São Paulo: Érica, 2002.
  - Capítulo 7
- ARNOLD, K., Programando em Java, São Paulo: Makron Books, 353 p., 1997.
  - Capítulo 3 e 4
- Universidade de Brasília (UnB FGA)
  - https://cae.ucb.br/conteudo/unbfga
    (escolha a disciplina **Orientação a Objetos** no menu superior)