Lição 1

Revisão dos Conceitos Básicos em Java

Objetivos

Ao final desta lição, o estudante será capaz de:

- Explicar e usar os conceitos básicos de orientação a objetos em seus códigos
  - Classes, objetos, atributos, métodos e construtores
- Descrever conceitos avançados de orientação a objetos e aplicá-los na codificação
  - Pacote, encapsulamento, abstração, herança, polimorfismo e interface
- Descrever e utilizar as palavras-chaves: this, super, final e static
- Diferenciar entre polimorfismo por overloading e override

Conceitos da Orientação a Objeto

- Focada em classes e objetos baseados em cenários do mundo real
- Enfatiza estado, comportamento e interação dos Objetos
- Vantagens:
  - Desenvolvimento rápido
  - Aumento da qualidade
  - Facilita manutenção e as mudanças
  - Aumenta a reutilização de software

Conceitos da Orientação a Objeto

- Classe
  - Permite definir novos tipos de dados
- Objeto
  - Entidade que tem um estado, um comportamento e uma identidade
  - Construído a partir de uma classe
- Atributo
  - Elemento que representa os dados de um objeto
  - Armazena informações sobre o objeto

Conceitos da Orientação a Objeto

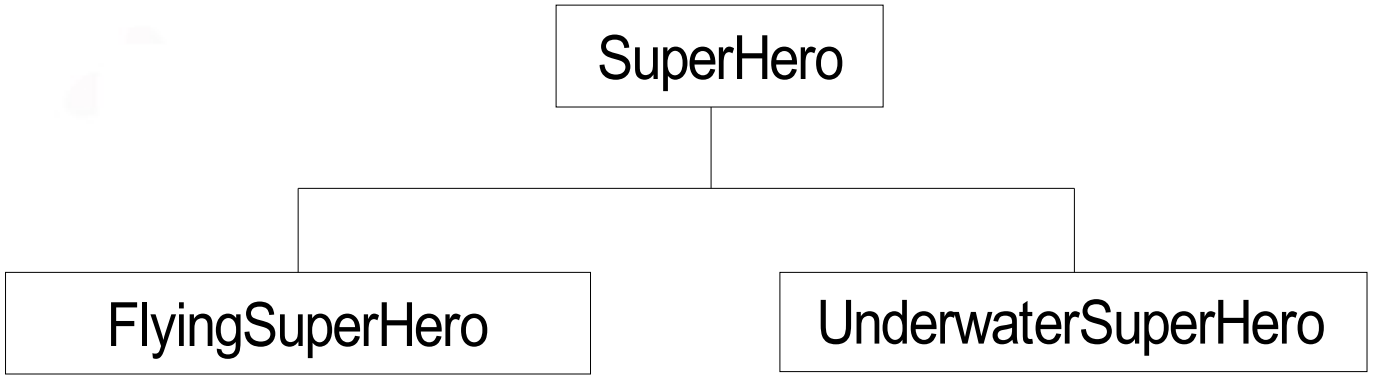
- Método
  - Descreve o comportamento do objeto
- Construtor
  - Para construir e inicializar um novo objeto

Conceitos da Orientação a Objeto

- Pacote
  - Refere-se a um grupo de classes e/ou sub pacotes
  - Estrutura semelhante ao de diretórios
- Encapsulamento
  - Principal forma de implementar informações
- Abstração
  - Ignora aspectos não importantes ao objeto e concentra-se nos que são

Conceitos da Orientação a Objeto

- Herança
  - Relacionamento entre classes onde uma classe é a super-classe ou uma classe pai de outra
  - Refere-se às propriedades e aos comportamentos recebidos de um ancestral
  - Conhecido como um relacionamento “é-um”



Conceitos da Orientação a Objeto

- Polimorfismo
  - Capacidade que um método possui de assumir diferentes formas

- Interface
  - Forma de contrato contendo uma coleção de métodos e atributos não modificáveis
  - Classes que implementam as interfaces, devem seguir as regras de contratação

Estrutura de codificação Java: Declarando uma classe Java

- Sintaxe

```
<declaraçãoClasse> ::=
   <modificador> class <nome> {
      <declaraçãoAtributo>*
      <declaraçãoConstructor>*
      <declaraçãoMétodo>*
   }
```

Estrutura de codificação Java: Declarando uma classe Java

```
class SuperHero {
    String superPowers[];
    void setSuperPowers(String superPowers[])  {
        this.superPowers = superPowers;
    }
    void printSuperPowers() {
        for (int i = 0; i < superPowers.length; i++) {
            System.out.println(superPowers[i]);
        }
    }
}
```

Estrutura de codificação Java: Declarando atributos

- Sintaxe:

```
<declaraçãoAtributo> ::=
   <modificador> <tipo> <nome> [= <valorPadrão>];

<tipo> ::=
   byte | short | int | long | char | float |
double  | boolean | <classeQualquer>
```

Estrutura de codificação Java: Declarando atributos

```
public class AttributeDemo {
    private String studNum;
    public boolean graduating = false;
    protected float unitsTaken = 0.0f;
    String college;
}
```

Estrutura de codificação Java: Declarando métodos

- Sintaxe:

```
<declaraçãoMétodo> ::=
   <modificador> <tipoRetorno> <nome>(<argumento>*) {
      <instrução>*
   }
<argumento> ::=
   <tipoArgumento> <nomeArgumento>[,]
```

Estrutura de codificação Java: Declarando métodos

```
class MethodDemo {
    int data;
    int getData() {
        return data;
    }
    void setData(int data) {
        this.data = data;
    }
    void setMaxData(int data1, int data2) {
        data = (data1>data2)? data1 : data2;
    }
}
```

Estrutura de codificação Java: Declarando um construtor

- Sintaxe:

```
<declaraçãoConstrutor> ::=
    <modificador> <NomeClasse> (<argumento>*) {
        <instrução>*
    }
```

- Construtor padrão

Estrutura de codificação Java: Declarando um construtor

```
class ConstructorDemo {
   private int data;
   public ConstructorDemo() {
      data = 100;
   }
   ConstructorDemo(int data) {
      this.data = data;
   }
}
```

Estrutura de codificação Java: Instanciando uma classe

- Sintaxe:

```
new <NomeConstrutor>(<argumentos>)
```

- Exemplo:

```
class ConstructObj {
    int data;
    ConstructObj() {
    }
    public static void main(String args[]) {
       ConstructObj obj = new ConstructObj();
    }
}
```

Estrutura de codificação Java: Acessando membros de um objeto

- Notação Ponto:

```
<objeto>.<elemento>
```

- Exemplo:

```
String myString = new String("My String");
System.out.println("Length: " + myString.length());
```

Estrutura de codificação Java: Pacote

- Sintaxe que indica que o código pertence ao Pacote:

```
<declaraçãoPacote> ::=
   package <nomePacote>;
```

- Simtaxe para importar outros pacotes:

```
<declaraçãoImportação> ::=
   import <nomePacote.elementoAcessado>;
```

- Formato do código Fonte:

```
[<declaraçãoPacote>]
<declaraçãoImportação>*
<declaraçãoClasse>+
```

Estrutura de codificação Java: Pacote

```
package registration.reports;

import registration.processing.*;
import java.util.List;

class MyClass {
   /* Detalhes de MyClass */
}
```

Estrutura de codificação Java: Modificadores de acesso

	private	**default**	*protected*	*public*
Mesma Classe	sim	sim	sim	sim
Mesmo pacote	-	sim	sim	sim
Pacotes diferentes (subclasse)	-	-	sim	sim
Pacotes diferentes (não-subclasse)	-	-	-	sim

Estrutura de codificação Java: Encapsulamento

- Utiliza o modificador *private* para proteger um atributo

```
class Encapsulation {
   private int secret;
   public boolean setSecret(int secret) {
      if (secret < 1 || secret > 100)
         return false;
      this.secret = secret;
      return true;
   }
   public getSecret() {
      return secret;
   }
}
```

Estrutura de codificação Java: Herança

- Criando uma classe filha ou sub-classe
  - Utilizar a palavra-chave *extends* na declaração da classe

```
class <NomeClasseFilha> extends <NomeClassePai>
```

- A classe pode estender apenas uma classe pai (super-classe)

Estrutura de codificação Java: Herança

```
import java.awt.*;

class Point {
    int x;
    int y;
}

class ColoredPoint extends Point {
    Color color;
}
```

Estrutura de codificação Java: Override de métodos

- A subclasse define um método cuja assinatura é idêntica ao do método definido na superclasse

- Assinatura de um método
  - Tipo de retorno
  - Nome do método
  - Lista de argumentos do método

Estrutura de codificação Java: Override de métodos

```
class Superclass {
   void display(int n) {
      System.out.println("super: " + n);
   }
}
class Subclass extends Superclass {
   void display(int k) {
      System.out.println("sub: " + k);
   }
}
class OverrideDemo {
   public static void main(String args[]) {
      Subclass SubObj = new Subclass();
      Superclass SuperObj = SubObj;
      SubObj.display(3);
      ((Superclass)SubObj).display(4);
   }
}
```

Estrutura de codificação Java: Override de métodos

- Versão do método chamado
  - Baseado no tipo de dado atual do objeto que invocou o método
- Modificador de acesso deve ser igual ou menos restritivo

Estrutura de codificação Java: Override de métodos

```
class Superclass {
   void overriddenMethod() {
   }
}
class Subclass1 extends Superclass {
   public void overriddenMethod() {
   }
}
class Subclass2 extends Superclass {
   void overriddenMethod() {
   }
}
class Subclass3 extends Superclass {
   protected void overriddenMethod() {
   }
}
class Subclass4 extends Superclass {
   private void overriddenMethod() {
   }
}
```

Estrutura de codificação Java: Métodos e classes abstratas

- Sintaxe:

```
abstract <modificador> <tipoRetorno>
  <nome>(<argumento>*);
```

- Classe que contém um método *abstract* deve ser declarada *abstract*

```
abstract class <Nome> {
   /* construtores, campos e métodos */
}
```

Estrutura de codificação Java: Métodos e classes abstratas

- Palavra reservada *abstract* não pode ser utilizada:
  - Construtor
  - Método estático
- *Classes abstract* não podem ser instanciadas
- Classe que estende uma classe abstract:
  - Deve implementar todos os métodos *abstract*
  - Caso algum método não seja implementado, a classe deve ser declarada *abstract*

Estrutura de codificação Java: Métodos e classes abstratas

```
abstract class SuperHero {
   abstract void displayPower();
}
class Superman extends SuperHero {
   void displayPower() {
      System.out.println("Fly...");
   }
}

class Spiderman extends SuperHero {
   void displayPower() {
     System.out.println("Fast...");
   }
}
```

Estrutura de codificação Java: Interface

- Sintaxe:

```
<declaraçãoInterface> ::=
    <modificador> interface <Nome> {
        <declaraçãoAtributo>*
        [<modificador> <tipoRetorno> <nome>
            (<argumentos>*);]*
    }
```

Estrutura de codificação Java: Interface

- Atributos da interface:
  - *public*, *static* e *final*
- Métodos:
  - *public*
- Implementando uma interface:
  - Palavra-chave *implements*
  - Implementar todos os métodos da interface
  - A classe pode implementar várias interfaces

Estrutura de codificação Java: Interface

```
interface MyInterface {
   void iMethod();
}

class MyClass1 implements MyInterface {
   public void iMethod() {
      System.out.println("Interface method.");
   }
   void myMethod() {
      System.out.println("Another method.");
   }
}
class MyClass2 implements MyInterface {
   public void iMethod() {
      System.out.println("Another implementation.");
   }
}
```

Estrutura de codificação Java: Palavra-chave *this*

- Evitar a ambigüidade entre atributo local e do método

```
class ThisDemo1 {
    int data;
    void method(int data) {
        this.data = data;
    }
}
```

Estrutura de codificação Java: Palavra-chave *this*

- Referenciar-se a um objeto que invoca um método não-estático

```
class ThisDemo2 {
    int data;
    void method() {
        System.out.println(this.data);
    }
    void method2() {
        this.method();
    }
}
```

Estrutura de codificação Java: Palavra-chave *this*

- Overloading de construtores:
  - Métodos diferentes de uma classe que compartilham o mesmo nome
  - Lista de Parâmetros deve ser diferente

```
class MyClass {
    void myMeth() {}
    void myMeth(int i) {}
    void myMeth(int i, int j) {}
}
```

Estrutura de codificação Java: Palavra-chave *this*

- Referenciar-se a outros construtores

```
class ThisDemo3 {
    int data;
    ThisDemo3() {
        this(100);
    }
    ThisDemo3(int data) {
        this.data = data;
    }
}
```

- A instrução *this()* deve ser a primeira declaração do construtor

Estrutura de codificação Java: Palavra-chave *super*

- Relacionado a hierarquia
  - Invoca o construtor da super-classe
  - Pode ser usado como a palavra reservada *this* referindo-se a elementos da super-classe

- Chamando construtores da super-classe

- *super()*
  - Refere-se imediatamente à super-classe
  - Pode ser a primeira declaração do construtor da sub-classe

Estrutura de codificação Java: Palavra-chave *super*

```
class Person {
    String firstName;
    String lastName;
    Person(String fname, String lname) {
        firstName = fname;
        lastName = lname;
    }
}
class Student extends Person {
 String studNum;
 Student(String fname, String lname, String sNum) {
        super(fname, lname);
        studNum = sNum;
    }
}
```

Estrutura de codificação Java: Palavra-chave *super*

- Referindo-se a elementos da super-classe

```
class Superclass{
    int a;
    void display_a(){
        System.out.println("a = " + a);
    }
}
class Subclass extends Superclass {
    int a;
    void display_a(){
        System.out.println("a = " + a);
    }
    void set_super_a(int n){
        super.a = n;
    }
    void display_super_a(){
        super.display_a();
    }
}
```

Estrutura de codificação Java: Palavra-chave *static*

- Aplicada aos elementos de uma classe:
  - Atributos
  - Métodos
  - Classes internas

- Permite acessar elementos da classe, ou *static*, sem ter sido instanciada

- Atributos da classe
  - Comportamento igual ao atributo global
  - Podem ser acessados por todas as instâncias da classe

Estrutura de codificação Java: Palavra-chave *static*

- Métodos da classe
  - Podem ser invocados sem criar um objeto dessa classe
  - Podem acessar somente elementos estáticos da classe
  - Não podem referir-se a *this* ou *super*

- *Blocos estáticos*
  - Chamados apenas uma única vez, quando a classe é carregada
  - Para inicializar atributos de classe

Estrutura de codificação Java: Palavra-chave *static*

```
class Demo {
    static int a = 0;
    static void staticMethod(int i) {
        System.out.println(i);
    }
    static {  //static block
        System.out.println("static block");
        a += 1;
    }
}
```

Estrutura de codificação Java: Palavra-chave *final*

- Aplicada para atributos, métodos e classes
- Restringir modificação
- Palavra reservada *final* pode ser colocada antes ou depois de outros modificadores
- **Atributo *final***

```
final int data = 10;
data++;
```

Estrutura de codificação Java: Palavra-chave *final*

- **Método *final***

```
public class MyClass {
   final void myMethod() {
   }
}
class ChildClass extends MyClass {
   void myMethod() {
   }
}
```

Estrutura de codificação Java: Palavra-chave *final*

- **Classe *final***

```
final public class MyClass {}
class WrongClass extends MyClass {}
```

Estrutura de codificação Java: Classes internas (Inner Class)

- Classe declarada dentro de outra classe
- Para acessar elementos da classe interna e necessário de uma instância da classe interna

```
innerObj.innerMember = 5;
```

Estrutura de codificação Java: Classes internas (Inner Class)

```
class Out {
    int outData;
    class In {
        void inMeth() {
            outData = 10;
        }
    }
}
```

Estrutura de codificação Java: Classes internas (Inner Class)

```
class OuterClass {
   int data = 5;
   class InnerClass {
      int data2 = 10;
      void method() {
         System.out.println(data);
         System.out.println(data2);
      }
   }
   public static void main(String args[]) {
      OuterClass oc = new OuterClass();
      InnerClass ic = oc.new InnerClass();
      System.out.println(oc.data);
      System.out.println(ic.data2);
      ic.method();
   }
}
```

Sumário

- Conceitos da Orientação a objeto
  - Design de orientação a objeto
  - Classe
  - Objeto
  - Atributo
  - Método
  - Construtor
  - Pacote
  - Encapsulamento
  - Abstração
  - Herança
  - Polimorfismo
  - Interface

Sumário

- Estrutura de codificação Java
  - Declarando uma classe Java
  - Declarando Atributos
  - Declarando métodos
  - Declarando um construtor
  - Instanciando uma classe
  - Acessando elementos de um objeto
  - Pacotes
  - Modificadores de Acesso
  - Encapsulamento
  - Herança
  - Override de métodos
  - Métodos e classes abstratas
  - Interface
  - Palavra-chave *this*
  - Palavra-chave *super*
  - Palavra-chave *static*
  - Palavra-chave *final*
  - Classes internas

Parceiros

- **Os seguintes parceiros tornaram JEDI™ possível em Língua Portuguesa:**