Lição 10

Criando nossas classes

Objetivos

Ao final da lição, o estudante deverá estar apto a:

- Criar nossas classes
- Declarar atributos e métodos para as classes
- Usar o objeto **this** para acessar dados de instância
- Utilizar overloading de métodos
- Importar e criar pacotes
- Utilizar modificadores de acesso para controlar o acesso aos elementos de uma classe

Definindo as próprias classes

- Algumas observações devem ser feitas quanto à sintaxe definida para esta seção:

*****	indica que pode haver nenhuma ou diversas ocorrências na linha em que for aplicada
<descrição>	indica que você deve substituir este trecho por um certo valor, ao invés de digitá-lo tal como está
[ ]	indica que esta parte é opcional

Definindo nossas próprias classes

```
<modificador>* class <nome> {
    <declaraçãoDoAtributo>*
    <declaraçãoDoConstrutor>*
    <declaraçãoDoMétodo>*
}
```

Exemplo

```
public class StudentRecord {
      // adicionaremos mais código aqui
}
```

Declarando Atributos

```
<modificador>* <tipo> <nome> [= <valorInicial>];
```

Atributos de Objeto

```
public class StudentRecord {
    private String name;
    private String address;
    private int    age;
    private double mathGrade;
    private double englishGrade;
    private double scienceGrade;
}
```

Atributos de Classe (Estáticas)

```
public class StudentRecord {
    private static int studentCount;
}
```

- usamos a palavra-chave static para indicar que um atributo estático

Declarando Métodos

```
<modificador>* <tipoRetorno> <nome>(<argumento>*) {
    <instrução>*
}
```

- Argumentos são separados por vírgulas:

```
<tipoArgumento> <nomeArgumento>
```

Métodos Acessores

- Usados para ler valores de atributos de classe ou de objeto
- Escritos como:

```
get<NomeDoAtributo>
```

- Retorna o valor do atributo

Exemplo

```
public class StudentRecord {
    public String getName() {
        return name;
    }
    public double getAverage() {
        double result = 0;
        result=(mathGrade+englishGrade+scienceGrade)/3;
        return result;
    }
}
```

Métodos Modificadores

- Utilizados para modificar os valores dos atributos de classe ou de objeto
- Escritos como:

```
set<NomeDoAtributo>
```

- Recebe o valor do atributo

Exemplo

```
public class StudentRecord {
    public void setName(String temp) {
        name = temp;
    }
}
```

Múltiplos Comandos return

- Desde que eles não pertençam ao mesmo bloco
- Pode-se utilizar constantes para retornar valores, ao invés de atributos

Exemplo

```
public String getNumberInWords(int num) {
    String defaultNum = "zero";
    if (num == 1) {
       return "one";
    } else if (num == 2) {
       return "two";
    }
    return defaultNum;
}
```

Métodos Estáticos

```
public class StudentRecord {
    private static int studentCount;
    public static int getStudentCount() {
        return studentCount;
    }
}
```

Exemplo de Utilização da Classe

```
public class StudentRecordExample {
   public static void main(String[] args){
      StudentRecordannaRecord = new StudentRecord();
      StudentRecordbeahRecord = new StudentRecord();
      StudentRecordcrisRecord = new StudentRecord();

      annaRecord.setName("Anna");
      beahRecord.setName("Beah");
      crisRecord.setName("Cris");

      System.out.println(annaRecord.getName());

      System.out.println(
         "Count=" + StudentRecord.getStudentCount());
   }
}
```

this

- Utilizado para acessar atributos ou métodos de objeto

```
this.<nomeDoAtributo>
```

- Exemplo:

```
public void setAge(int age) {
    this.age = age;
}
```

Overloading de Métodos

- Permite que um método com o mesmo nome e diferentes argumentos, possa ter implementações diferentes e retornar valores de diferentes tipos
- Pode ser usado quando a mesma operação tem implementações diferentes
- Propriedades:
  - mesmo nome
  - argumentos diferentes
  - tipo do retorno pode ser igual ou diferente

Exemplo

```
public void print() {
    System.out.println("Name:" + name);
    System.out.println("Address:" + address);
    System.out.println("Age:" + age);
}
public void print(double eGrade, double mGrade, double sGrade) {
    System.out.println("Name:" + name);
    System.out.println("Math Grade:" + mGrade);
     System.out.println("English Grade:" + eGrade);
    System.out.println("Science Grade:" + sGrade);
}
```

Exemplo

```
public static void main(String[] args) {
    StudentRecord annaRecord =
        new StudentRecord();

    annaRecord.setName("Anna");
    annaRecord.setAddress("Philippines");
    annaRecord.setAge(15);
    annaRecord.setMathGrade(80);
    annaRecord.setEnglishGrade(95.5);
    annaRecord.setScienceGrade(100);

    annaRecord.print();

    annaRecord.print(
        annaRecord.getEnglishGrade(),
        annaRecord.getMathGrade(),
        annaRecord.getScienceGrade());
}
```

Saída

- teremos a saída para a primeira chamada ao **print**:

```
Name:Anna
Address:Philippines
Age:15
```

- teremos a seguinte saída para a segunda chamada ao **print**:

```
Name:Anna
Math Grade:80.0
English Grade:95.5
Science Grade:100.0
```

Construtores

- Importantes na criação de um objeto
- É um método onde são colocadas todas as inicializações
- Possuem o mesmo nome da classe
- Não retornam valor
- Executados automaticamente na utilização do operador **new** durante a instanciação da classe

Construtores

```
[modificador] <nomeClasse> (<argumento>*) {
    <instrução>*
}
```

Construtor Padrão (Default)

- Público e sem argumentos
- Se não for definido um construtor para a classe, é assumido o construtor padrão

```
public StudentRecord() {
}
```

Overloading de Construtores

```
public StudentRecord(){
}
public StudentRecord(String name){
    this.name = name;
}
public StudentRecord(String name, String address){
    this.name = name;
    this.address = address;
}
public StudentRecord(double mGrade, double eGrade,
        double sGrade){
    mathGrade = mGrade;
    englishGrade = eGrade;
    scienceGrade = sGrade;
}
```

Utilizando Construtores

```
public static void main(String[] args) {
    StudentRecord annaRecord = new StudentRecord("Anna");
    StudentRecord beahRecord = new StudentRecord("Beah",
        "Philippines");
    StudentRecord crisRecord = new StudentRecord(80,90,100);
}
```

Utilizando o this()

- Podem ser cruzadas, o que significa que você pode chamar um construtor de dentro de outro construtor
- Deve sempre ocorrer na primeira linha de instrução
- Utilizado para a chamada a um Construtor

Exemplo

```
public StudentRecord() {
    this("some string");
}
public StudentRecord(String temp) {
    this.name = temp;
}
public static void main(String[] args) {
    StudentRecord annaRecord = new StudentRecord();
}
```

Pacotes

- É utilizado no Java para agrupar classes e interfaces relacionadas em uma única unidade

- Oferece um mecanismo conveniente para o gerenciamento de um grupo grande de classes e interfaces, e evita conflitos de nomes

Importando Pacotes

- Para utilizar classes externas ao pacote atual
- Por padrão, todos os programas Java importam o pacote **java.lang**
- A sintaxe para importar pacotes é:

```
import <nomeDoPacote>.<nomeDaClasse>;
```

- Exemplo:

```
import java.awt.Color;
import java.awt.*;
```

Criando Pacotes

- Para criar nossos pacotes, escrevemos:

```
package <nomePacote>;
```

- Pacotes também podem ser aninhados. Neste caso, o interpretador espera que a estrutura de diretórios contendo as classes combinem com a hierarquia dos pacotes

Exemplo

```
package schoolClasses;

public class StudentRecord {
    private String   name;
    private String   address;
    private int      age;
    :
```

Definindo a CLASSPATH

- Estrutura de Diretórios:

```
C:\
    schoolClasses\
        StudentRecord.java
```

- Precisamos definir a *classpath* para apontar para este diretório.

Definindo a CLASSPATH

```
C:\schoolClasses>javac StudentRecord.java

C:\schoolClasses>java StudentRecord
Exception in thread "main" java.lang.NoClassDefFoundError:
StudentRecord (wrong name: schoolClasses/StudentRecord)
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(Unknown Source)
        at java.security.SecureClassLoader.defineClass(Unknown Source)
        at java.net.URLClassLoader.defineClass(Unknown Source)
        at java.net.URLClassLoader.access$100(Unknown Source)
        at java.net.URLClassLoader$1.run(Unknown Source)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(Unknown Source)
        at java.lang.ClassLoader.loadClass(Unknown Source)
        at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
        at java.lang.ClassLoader.loadClass(Unknown Source)
        at java.lang.ClassLoader.loadClassInternal(Unknown Source)
```

Definindo a CLASSPATH

- Definir a **classpath** no Windows:

```
C:\schoolClasses>set classpath=C:\
```

- Poderemos executar a nossa classe em qualquer lugar:

```
C:\schoolClasses>java schoolClasses.StudentRecord
```

Definindo a CLASSPATH

- Para sistemas baseados no Unix e supondo que as classes estejam no diretório /usr/local/myClasses:

```
export classpath=/usr/local/myClasses
```

Definindo a CLASSPATH

- Em qualquer lugar
- Mais de um local de pesquisa, separar por:
  `;` (no Windows)
  `:` (nos sistemas baseados em Unix)
- Para sistemas baseados no Windows:

```
set classpath=C:\myClasses;D:\;E:\MyPrograms\Java
```

- Para sistemas baseados no Unix:

```
export classpath=/usr/local/java:/usr/myClasses
```

Modificadores de Acesso

- Há quatro diferentes tipos de modificadores de acesso:
  - public
  - private
  - protected
  - default

- **public**, **protected** e **private** são escritos explicitamente na instrução para indicar o tipo de acesso

- **default** não deve ser escrito

Acesso Padrão

- Especifica que os elementos da classe são acessíveis somente aos métodos internos da classe e às suas subclasses

- Não há palavra-chave para o modificador **default**; sendo aplicado na ausência de um modificador de acesso

Exemplo

```
public class StudentRecord {
   int name;

   String getName(){
      return name;
   }
}
```

Acesso Público

- Especifica que os elementos da classe são acessíveis seja internamente e externamente à classe
- Qualquer objeto que interage com a classe pode ter acesso aos elementos públicos da classe
- Palavra-chave: **public**

Exemplo

```
public class StudentRecord {
   public int name;

   public String getName(){
      return name;
   }
}
```

Acesso Protegido

- Especifica que somente classes no mesmo pacote podem ter acesso aos atributos e métodos da classe

- Palavra-chave: **protected**

Exemplo

```
public class StudentRecord {
   protected int name;

   protected String getName(){
      return name;
   }
}
```

Acesso Particular

- Especifica que os elementos da classe são acessíveis apenas pela classe que os definiram
- Palavra-chave: **private**

Exemplo

```
public class StudentRecord {
   private int name;

   private String getName(){
      return name;
   }
}
```

Sumário

- Definindo nossas classes
- Declarando Atributos (de objeto e de classe)
- Declarando Métodos (acessor, modificador, estático)
- Valores de retorno e múltiplos comandos **return**
- O objeto **this**
- Overloading de Métodos
- Construtores (padrão, overloading, **this()**)
- Pacotes
- Modificadores de Acesso (default, **public**, **private**, **protected**)

Parceiros

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