Programação 2

Classes e objetos: um exame mais profundo

Prof. Domingo Santos

domingos.santos@upe.br

UNIVERSIDADE DE PERNAMBUCO

Resumo

- Estudo de caso da classe Time
  - Construtores Sobrecarregados
  - Composição
  - Garbage Collection
  - Variáveis Static
  - Variáveis de instância final

Considerações iniciais

Uma classe que manipula datas (hora, minuto e segundos):

- Lidando com as exceções
- palavra-chave this
- sobrecarga de métodos
- utização de metodos get e set
- Variáveis estáticas e “final”

Classe Time1

- As variáveis de instância private int:
  - hour: formato de 0 até 23
  - minute e second: formado de 0 até 59

Classe Time1

Definindo classe Time1 versão inicial - parte 1

```
public class Time1
{
   private int hour;
   private int minute;
   private int second;

   public void setTime(int hour, int minute, int second)
   {
       // valida hora, minuto e segundo
       if (hour < 0 || hour >= 24 || minute < 0 || minute >= 60 || second < 0 || second >= 60

           throw new IllegalArgumentException("hour, minute and/or second was out of range");
       }
       this.hour = hour;
       this.minute = minute;
       this.second = second;
    }
  . . .
}
```

Classe Time1

Definindo classe Time1 versão inicial - parte 2

```
public class Time1
{
   . . .

  // valida e configura a hora
   public void setHour(int hour)
   {
          if (hour < 0 || hour >= 24)
            throw new IllegalArgumentException("hour must be 0-23");
          this.hour = hour;
   }

   // valida e configura os minutos
   public void setMinute(int minute)
   {
      if (minute < 0 || minute >= 60)
            throw new IllegalArgumentException("minute must be 0-59");

      this.minute = minute;
   }

   // valida e configura os segundos
   public void setSecond(int second)
   {
          if (second < 0 || second >= 60)
            throw new IllegalArgumentException("second must be 0-59");

          this.second = second;
   }
  . . .
}
```

Classe Time1

Definindo classe Time1 versão inicial - parte 3

```
public class Time1
{
  . . .
   public int getHour()
   {
      return hour;
   }
   // obtém valor dos minutos
   public int getMinute()
   {
      return minute;
   }
   // obtém valor dos segundos
   public int getSecond()
   {
      return second;
   }

  . . .
}
```

Classe Time1

Definindo classe Time1 versão inicial - parte 4

```
public class Time1
{
   . . .
    // converte em String no formato de data/hora universal (HH:MM:SS)
    public String toUniversalString ()
    {
        return String.format("%02d:%02d:%02d", hour, minute, second);
    }
    // converte em String no formato padrão de data/hora (H:MM:SS AM ou PM)
    public String toString ()
    {
      int hour;

      String am_pm = "PM";
      if (this.hour==0 || this.hour==12){
            hour = 12;
      }else{
            hour = this.hour % 12;
      }

      if (this.hour < 12){
            am_pm = "AM";
      }
      return String.format("%d:%02d:%02d %s", hour, minute, second, am_pm);
    }
}
```

Classe Time1

Definindo classe Time1Test versão inicial - parte 1

```
public class Time1Test {
     public static void main(String[] args) {
               . . .
     }
     // exibe um objeto Time1 nos formatos de 24 horas e 12 horas
     private static void displayTime(String header, Time1 t)
     {
          System.out.printf("%s%nUniversal time: %s%nStandard time: %s%n",
          header, t.toUniversalString(),t.toString());
     }
}
```

Classe Time1

Definindo classe Time1Test versão inicial - parte 2

```
public class Time1Test {
      public static void main(String[] args) {
            Time1 time = new Time1(); // invoca o construtor Time1

            // gera saída de representações de string da data/hora
            displayTime("After time object is created", time);
            System.out.println();

            // altera a data/hora e gera saída da data/hora atualizada
            time.setTime(13, 27, 6);
            displayTime("After calling setTime", time);
            System.out.println();

            // tenta definir data/hora com valores inválidos
            try
            {
                  time.setTime(99, 99, 99); // fora do intervalo
            }
            catch (IllegalArgumentException e)
            {
                  System.out.printf("Exception: %s%n%n", e.getMessage());
            }
            // exibe a data/hora após uma tentativa de definir valores inválido
            displayTime("After calling setTime with invalid values", time);
      }
      . . .
}
```

```
After time object is created
Universal time: 00:00:00
Standard time: 12:00:00 AM

After calling setTime
Universal time: 13:27:06
Standard time: 1:27:06 PM

Exception: hour, minute and/or second
was out of range

After calling setTime with invalid
values

Universal time: 13:27:06
Standard time: 1:27:06 PM
```

Classe Time1

Considerando a classe Time1, é possível realizar a seguinte operação?

```
Time1 time = new Time1();
time.hour = 7;
time.minute = 15;
time.second = 30;
```

Não ! As variáveis são privadas. Os clientes não precisam se preocupar com a forma como a classe realiza suas tarefas.

Caso seja necessário definir as variáveis separadas devemos usar os métodos Get e Set disponíveis

Classe Time1

Referenciando membros do objeto atual com a referência this

Caso 1

```
public String toUniversalString()
{
    return String.format("%02d:%02d:%02d", hour, minute, second);
}
```

Caso 2

```
public String toUniversalString()
{
    return String.format("%02d:%02d:%02d", this.hour, this.minute, this.second);
}
```

Os casos acima possuem diferenças práticas?

nesse caso Não ! Os métodos da classe “enxergam” as variáveis globais. O “this” deixa claro que qual variável está sendo utilizada

Classe Time1

Referenciando membros do objeto atual com a referência this

Caso 3

```
public static String toUniversalString()
{
    return String.format("%02d:%02d:%02d", hour, minute, second);
}
```

Da forma como o método está especificado, seria possível que o mesmo fosse estático?

Não ! Métodos estáticos por definição são realizam tarefas que não depende de um objeto

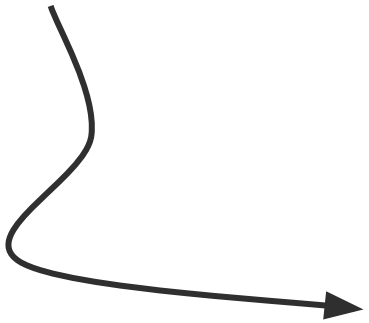
Classe Time1 com Construtores Sobrecarregados

```
public class Time1
{
   . . .

   public Time1(int hour, int minute, int second)
   {
      // valida hora, minuto e segundo
       if (hour < 0 || hour >= 24 || minute < 0 || minute >= 60 || second < 0 || second >= 60){

           throw new IllegalArgumentException("hour, minute and/or second was out of range");
       }

          this.hour = hour;
          this.minute = minute;
          this.second = second;
   }
  . . .
}
```



```
Time1 time = new Time1(0, 0, 0);
```

Classe Time1 com Construtores Sobrecarregados

```
public class Time1
{

   private int hour; // 0 - 23
   private int minute; // 0 - 59
   private int second; // 0 - 59

   public Time1()
   {
     this(0, 0, 0);
    // invoca o construtor com três argumentos
   }

   public Time1(int hour)
   {
     this(hour, 0, 0);
   }

   public Time1(int hour, int minute)
   {
     this(hour, minute, 0);
   }

  . . .
}
```

```
Time1 time = new Time1();
```

```
Time1 time = new Time1(0);
```

```
Time1 time = new Time1(0, 0);
```

Classe Time disponível na página 255 do livro de Deitel

Composição

Uma classe pode ter referências a objetos de outras classes como membros.

Por exemplo:

- Classe empregado possui as seguintes variáveis:
  - String primeiroNome;
  - String últimoNome
  - Time1 horaEntrada
  - Time1 horaSaida

Composição

Implementação classe empregado

```
public class Employee {
     private String firstName;
     private String lastName;
     private Time1 enterHour;
     private Time1 exitHour;

     public Employee(String firstName, String lastName, Time1 enterHour, Time1 exitHour){
               this.firstName = firstName;
               this.lastName = lastName;
               this.enterHour = enterHour;
               this.exitHour = exitHour;
     }

     // converte Employee em formato de String
     public String toString()
     {
          return String.format("%s, %s Hired: %s Birthday: %s" , lastName, firstName, enterHour, exitH
     }

}
```

Composição

Chamando a classe “Empregado”

```
public class EmployeeTest {
     public static void main(String[] args){

          Time1 enterHour = new Time1(8, 0, 0);
          Time1 exitHour = new Time1(16, 30, 0);
          Employee employee = new Employee("Vinsmoke", "Sanji", enterHour, exitHour);

          System.out.println(employee);
     }
}
```

UNIVERSIDADE DE PERNAMBUCO

Nota: Garbage Collection

- É um mecanismo automático da JVM que gerencia a alocação e liberação de memória para objetos.
- Objetos que não são mais referenciados pelo programa são identificados e removidos da memória pelo garbage collector.
- Assim, não ocorre vazamentos de memória que são comuns em outras linguagens como C e C++
- Normalmente o programador não precisa realizar ações sobre o Garbage Collection

UNIVERSIDADE DE PERNAMBUCO

Variáveis Static (variável de classe)

Implementação classe empregado. Considerando a contagem de chamadas totais da classe (monitoramento)

```
public class Employee {
      private String firstName;
      private String lastName;
      private Time1 enterHour;
      private Time1 exitHour;
      private static int count = 0;

      public Employee(String firstName, String lastName, Time1 enterHour, Time1 exitHour){
                  this.firstName = firstName;
                  this.lastName = lastName;
                  this.enterHour = enterHour;
                  this.exitHour = exitHour;
                  this.count++;
      }

      // converte Employee em formato de String
      public String toString()
      {
            return String.format("%s, %s Entrada: %s Saída: %s, count: %d" , lastName, firstName,
enterHour.toString(), exitHour.toString(), count);
      }

}
```

UNIVERSIDADE DE PERNAMBUCO

Variáveis Static (variável de classe)

Chamando nova classe empregado

```
public class EmployeeTest {
     public static void main(String[] args){

          Time1 enterHour1 = new Time1(8, 0, 0);
          Time1 exitHour1 = new Time1(16, 30, 0);
          Employee employee1 = new Employee("Vinsmoke", "Sanji", enterHour1, exitHour1);

          System.out.println(employee1);

          Time1 enterHour2 = new Time1(7, 0, 0);
          Time1 exitHour2 = new Time1(17, 45, 0);
          Employee employee2 = new Employee("Vinsmoke", "Reiju", enterHour2, exitHour2);

          System.out.println(employee2);
          System.out.println();
          System.out.println(employee1);
     }
}
```

UNIVERSIDADE DE PERNAMBUCO

Variáveis de instância final

- O princípio do menor privilégio
- declara que deve ser concedido ao código somente a quantidade de privilégio e acesso que ele precisa para realizar sua tarefa designada
- Isso torna seus programas mais robustos evitando que o código modifique acidentalmente (ou maliciosamente) os valores das variáveis e chame métodos que não deveriam estar acessíveis.
- Palavra-chave final para especificar o fato de que uma variável **não é modificável**
- Constante
- Exemplo: private final int INCREMENT;
- Essas variáveis podem ser inicializadas quando elas são declaradas. Se não forem, elas devem ser inicializadas em cada construtor da classe.

Variáveis de instância final

Exemplo de uso

```
package methods;
public class Pessoa {
   private final String nome;
   private final int idade;
   private final int IDADE_MARIODADE=18;

   public Pessoa(String nome, int idade) {
       this.nome = nome;
       this.idade = idade;
   }

   public String getNome() {
       return nome;
   }

   public int getIdade() {
       return idade;
   }

   public boolean isMaior(){
     if (this.idade>=this.IDADE_MARIODADE) {
          return true;
     }

     return false;
   }
}
```

Variáveis de instância final

Prática, crie a classe Carro, com os seguintes

- Atributos:
  - proprietario tipo Proprietario:
  - ano tipo int
  - placa tipo String
- Métodos:
  - gets e sets dos atributos
  - toString
- Possui construtor com todas variáveis

Classe Proprietario:

- Atributos:
  - nome tipo String
  - idade tipo int
- Métodos:
  - gets e sets
  - toString
- Possui construtor com todas variáveis

Na classe main, crie o objeto Carro qualquer e exiba na tela o retorno do método toString