Tópicos

Estudo da Classe Time

- Declaração da classe *Time1*:
  - Criação de uma classe que representa um horário);
  - Quais as informações básicas de um horário?

Estudo da Classe Time

- Declaração da classe *Time1*:
  - Criação de uma classe que representa um horário);
  - Quais as informações básicas de um horário?
    - Horas, minutos e segundos.

```
//Time1.java
public class Time1 {
    int hour, minute, second;
}
```

Modificadores de acesso

```
//Time1Test.java
public class Time1Test {
    public static void main(String[] args) {
        // Cria e inicializa um objeto Time1
        // utilizando construtor padrão
        Time1 time = new Time1();
        time.hour = 13;
        time.minute = 27;
        time.second = 6;

        System.out.printf("Hora atual -> %d:%d%d",
        time.hour, time.minute, time.second);
    }
}
```

Modificadores de acesso

```
//Time1Test.java
public class Time1Test {
    public static void main(String[] args) {
        // Cria e inicializa um objeto Time1
        // utilizando construtor padrão
        Time1 time = new Time1();
        time.hour = 13;
        time.minute = 27;
        time.second = 6;

        System.out.printf("Hora atual -> %d:%d%d",
        time.hour, time.minute, time.second);
    }
}
```

Problemas?

Modificadores de acesso

- Atributos *hour*, *minute* e *second* podem ser alterados sem obedecer ao padrão de um horário válido;
- Poderíamos cuidar que isso não acontece fazendo verificações na *TimeTest1.java*? Qual sua opinião sobre esse tipo de verificação?

Modificadores de acesso

- Atributos *hour*, *minute* e *second* podem ser alterados sem obedecer ao padrão de um horário válido;
- Poderíamos cuidar que isso não acontece fazendo verificações na *TimeTest1.java*? Qual sua opinião sobre esse tipo de verificação?
- Princípio da **coesão**: uma classe não deve assumir responsabilidades que não são suas;

Modificadores de acesso

- **Modificadores de acesso** controlam o acesso aos atributos e métodos de uma classe;

Modificadores de acesso

- **Modificadores de acesso** controlam o acesso aos atributos e métodos de uma classe;
- Modificadores de acesso que utilizaremos na disciplina:
  - **public**: método ou classe em questão pode ser invocado diretamente de um objeto. Ou seja, é público para quem desejar acessá-lo utilizando um objeto;
  - **private**: método ou classe em questão existirá no objeto, mas não poderá ser invocado diretamente a partir de um objeto. Está disponível para acesso apenas dentro dos métodos de um objeto.
- Vejamos um exemplo.

Estudo da Classe Time

- Redeclarando a classe *Time1*:

```
//Time1.java
public class Time1 {
    private int hour, minute, second;
}
```

- A partir de agora, atributos *hour*, *minute* e *second* não serão acessíveis através de um objeto. Ficou claro?

Estudo da Classe Time

- Redeclarando a classe *Time1*:

```
//Time1.java
public class Time1 {
    private int hour, minute, second;
}
```

- A partir de agora, atributos *hour*, *minute* e *second* não serão acessíveis através de um objeto. Ficou claro?
- Claro que não! Continuemos o exemplo.

Modificadores de acesso

```
//Time1Test.java
public class Time1Test {
    public static void main(String[] args) {
        // Cria e inicializa um objeto Time1
        // utilizando construtor padrão
        Time1 time = new Time1();
        time.hour = 13;
        time.minute = 27;
        time.second = 6;

        System.out.printf("Hora atual -> %d:%d%d",
        time.hour, time.minute, time.second);
    }
}
```

Modificadores de acesso

```
//Time1Test.java
public class Time1Test {
    public static void main(String[] args) {
        // Cria e inicializa um objeto Time1
        // utilizando construtor padrão
        Time1 time = new Time1();
        time.hour = 13;
        time.minute = 27;
        time.second = 6;

        System.out.printf("Hora atual -> %d:%d%d",
        time.hour, time.minute, time.second);
    }
}
```

- O código acima executará? Compilará?

Modificadores de acesso

```
//Time1Test.java
public class Time1Test {
    public static void main(String[] args) {
        // Cria e inicializa um objeto Time1
        // utilizando construtor padrão
        Time1 time = new Time1();
        time.hour = 13;
        time.minute = 27;
        time.second = 6;

        System.out.printf("Hora atual -> %d:%d%d",
        time.hour, time.minute, time.second);
    }
}
```

- O código acima executará? Compilará?
- Não, não e não! *hour*, *minute* e *second* agora são atributos *private*, portanto, não podem ser acessados através do objeto.

Setters e Getters

- Fornecem uma interface **pública** de acesso a atributos privados, controlando a sua alteração e retorno;
- Necessários, pois caso contrário, como alteraríamos e visualizaríamos atributos privados de um objeto?
- Vamos alterar a classe *Time1* e inserir métodos *setters* e *getters*.

Estudo da Classe Time

- Redeclarando a classe *Time1*:

```
//Time1.java
public class Time1 {
    private int hour, minute, second;
    public ??? setHour( ??? ) {
     ???
    }
    public ??? getHour( ??? ) {
     ???
    }
    public ??? setMinute( ??? ) {
     ???
    }
    public ??? getMinute( ??? ) {
     ???
    }
    public ??? setSecond( ??? ) {
     ???
    }
    public ??? getSecond( ??? ) {
     ???
    }
}
```

- Atributos *hour*, *minute* e *second* não serão acessíveis através de um objeto. Ficou claro?

Modificadores de acesso

- Claro que não! Vamos continuar o exemplo, reescrevendo a classe *Time1Test* utilizando os métodos *setters* e *getters*.

```
//Time1Test.java
public class Time1Test {
    public static void main(String[] args) {
        // Cria e inicializa um objeto Time1
        // utilizando construtor padrão
        Time1 time = new Time1();
        time.setHour(13);
        time.setMinute(27);
        time.setSecond(6);

        System.out.printf("Hora atual -> %d:%d:%d",
        time.getHour(), time.getMinute(), time.getSecond());
    }
}
```

Modificadores de acesso

- Claro que não! Vamos continuar o exemplo, reescrevendo a classe *Time1Test* utilizando os métodos *setters* e *getters*.

```
//Time1Test.java
public class Time1Test {
    public static void main(String[] args) {
        // Cria e inicializa um objeto Time1
        // utilizando construtor padrão
        Time1 time = new Time1();
        time.setHour(13);
        time.setMinute(27);
        time.setSecond(6);

        System.out.printf("Hora atual -> %d:%d:%d",
        time.getHour(), time.getMinute(), time.getSecond());
    }
}
```

- Vantagens e desvantagens?

Estudo da Classe Time

- Redeclarando a classe *Time1*:

```
//Time1.java
public class Time1 {
    private int hour, minute, second;
    public ??? setHour( ??? ) {
     ???
    }
    public ??? getHour( ??? ) {
     ???
    }
    public ??? setMinute( ??? ) {
     ???
    }
    public ??? getMinute( ??? ) {
     ???
    }
    public ??? setSecond( ??? ) {
     ???
    }
    public ??? getSecond( ??? ) {
     ???
    }
}
```

Estudo da Classe Time

- Redeclarando a classe *Time1*:

```
//Time1.java
public class Time1 {
    private int hour, minute, second;
    public void setHour( ??? ) {
     ???
    }
    public ??? getHour( ??? ) {
     ???
    }
    public ??? setMinute( ??? ) {
     ???
    }
    public ??? getMinute( ??? ) {
     ???
    }
    public ??? setSecond( ??? ) {
     ???
    }
    public ??? getSecond( ??? ) {
     ???
    }
}
```

Estudo da Classe Time

- Redeclarando a classe *Time1*:

```
//Time1.java
public class Time1 {
    private int hour, minute, second;
    public void setHour(int hour) {
     ???
    }
    public ??? getHour( ??? ) {
     ???
    }
    public ??? setMinute( ??? ) {
     ???
    }
    public ??? getMinute( ??? ) {
     ???
    }
    public ??? setSecond( ??? ) {
     ???
    }
    public ??? getSecond( ??? ) {
     ???
    }
}
```

Estudo da Classe Time

- Redeclarando a classe *Time1*:

```
//Time1.java
public class Time1 {
    private int hour, minute, second;
    public void setHour(int hour) {
     this.hour = hour
    }
    public ??? getHour( ??? ) {
     ???
    }
    public ??? setMinute( ??? ) {
     ???
    }
    public ??? getMinute( ??? ) {
     ???
    }
    public ??? setSecond( ??? ) {
     ???
    }
    public ??? getSecond( ??? ) {
     ???
    }
}
```

Estudo da Classe Time

- Redeclarando a classe *Time1*:

```
//Time1.java
public class Time1 {
    private int hour, minute, second;
    public void setHour(int hour) {
     this.hour = hour
    }
    public int getHour( ??? ) {
     ???
    }
    public ??? setMinute( ??? ) {
     ???
    }
    public ??? getMinute( ??? ) {
     ???
    }
    public ??? setSecond( ??? ) {
     ???
    }
    public ??? getSecond( ??? ) {
     ???
    }
}
```

Estudo da Classe Time

- Redeclarando a classe *Time1*:

```
//Time1.java
public class Time1 {
    private int hour, minute, second;
    public void setHour(int hour) {
     this.hour = hour
    }
    public int getHour() {
     ???
    }
    public ??? setMinute( ??? ) {
     ???
    }
    public ??? getMinute( ??? ) {
     ???
    }
    public ??? setSecond( ??? ) {
     ???
    }
    public ??? getSecond( ??? ) {
     ???
    }
}
```

Estudo da Classe Time

- Redeclarando a classe *Time1*:

```
//Time1.java
public class Time1 {
    private int hour, minute, second;
    public void setHour(int hour) {
     this.hour = hour
    }
    public int getHour() {
     return hour;
    }
    public ??? setMinute( ??? ) {
     ???
    }
    public ??? getMinute( ??? ) {
     ???
    }
    public ??? setSecond( ??? ) {
     ???
    }
    public ??? getSecond( ??? ) {
     ???
    }
}
```

Estudo da Classe Time

- Redeclarando a classe *Time1*:

```
//Time1.java
public class Time1 {
    private int hour, minute, second;
    public void setHour(int hour) {
     this.hour = hour
    }
    public int getHour() {
     return hour;
    }
    public void setMinute(int minute) {
     this.minute = minute;
    }
    public int getMinute() {
     return minute;
    }
    public void setSecond(int second) {
     this.second = second;
    }
    public int getSecond() {
     return second;
    }
}
```

Modificadores de acesso

- Qual era mesmo o nosso problema inicial ao utilizar um objeto do tipo *Time*?

```
//Time1Test.java
public class Time1Test {
    public static void main(String[] args) {
        // Cria e inicializa um objeto Time1
        // utilizando construtor padrão
        Time1 time = new Time1();
        ...
    }
}
```

Modificadores de acesso

- Qual era mesmo o nosso problema inicial ao utilizar um objeto do tipo *Time*?

```
//Time1Test.java
public class Time1Test {
    public static void main(String[] args) {
        // Cria e inicializa um objeto Time1
        // utilizando construtor padrão
        Time1 time = new Time1();
        // Não deveria ser permitido, e realmente agora não é
        time.hour = 43;
    }
}
```

Modificadores de acesso

- Qual era mesmo o nosso problema inicial ao utilizar um objeto do tipo *Time*?

```
//Time1Test.java
public class Time1Test {
    public static void main(String[] args) {
        // Cria e inicializa um objeto Time1
        // utilizando construtor padrão
        Time1 time = new Time1();
        // Mas o código abaixo é.
        time.setHour(43);
    }
}
```

Modificadores de acesso

- Qual era mesmo o nosso problema inicial ao utilizar um objeto do tipo *Time*?

```
//Time1Test.java
public class Time1Test {
    public static void main(String[] args) {
        // Cria e inicializa um objeto Time1
        // utilizando construtor padrão
        Time1 time = new Time1();
        // Mas o código abaixo é.
        time.setHour(43);
    }
}
```

- Os *setters* solucionaram nosso problema?

Modificadores de acesso

- Qual era mesmo o nosso problema inicial ao utilizar um objeto do tipo *Time*?

```
//Time1Test.java
public class Time1Test {
    public static void main(String[] args) {
        // Cria e inicializa um objeto Time1
        // utilizando construtor padrão
        Time1 time = new Time1();
        // Mas o código abaixo é.
        time.setHour(43);
    }
}
```

- Os *setters* solucionaram nosso problema?
- Da maneira como foram implementados, **não**.

Estudo da Classe Time

- Vamos mudar os setters da *Time1*:

```
//Time1.java
public class Time1 {
    private int hour, minute, second;
    public void setHour(int hour) {
     ???
    }
    public int getHour() {
     return hour;
    }
    public void setMinute(int minute) {
     ???
    }
    public int getMinute() {
     return minute;
    }
    public void setSecond(int second) {
     ???
    }
    public int getSecond() {
     return second;
    }
}
```

Estudo da Classe Time

- Vamos mudar os setters da *Time1*:

```
//Time1.java
public class Time1 {
    private int hour, minute, second;
    public void setHour(int hour) {
     this.hour =( (hour >= 0 && hour < 24) ? hour : 0 );
    }
    public int getHour() {
     return hour;
    }
    public void setMinute(int minute) {
     this.minute =( (minute >= 0 && minute < 60) ? minute : 0 );
    }
    public int getMinute() {
     return minute;
    }
    public void setSecond(int second) {
     this.second =( (second >= 0 && second < 60) ? second : 0 );
    }
    public int getSecond() {
     return second;
    }
}
```

Modificadores de acesso

- O que será impresso na tela?

```
//Time1Test.java
public class Time1Test {
    public static void main(String[] args) {
        // Cria e inicializa um objeto Time1
        // utilizando construtor padrão
        Time1 time = new Time1();
        // Mas o código abaixo é.
        time.setHour(43);
        System.out.printf("%d", time.getHour());
    }
}
```

Modificadores de acesso

- O que será impresso na tela?

```
//Time1Test.java
public class Time1Test {
    public static void main(String[] args) {
        // Cria e inicializa um objeto Time1
        // utilizando construtor padrão
        Time1 time = new Time1();
        // Mas o código abaixo é.
        time.setHour(43);
        System.out.printf("%d", time.getHour());
    }
}
```

- *Setters* mantêm os atributos com valores consistentes e a classe *Time1* ficou coesa!

Encapsulamento

- O uso de modificadores de acesso, juntamente com *getters* e *setters*, nos permite aplicar o conceito de **encapsulamento**;

Encapsulamento

- O uso de modificadores de acesso, juntamente com *getters* e *setters*, nos permite aplicar o conceito de **encapsulamento**;
- **Encapsulamento**: fornecer interfaces "seguras" de acesso aos componentes de sua classe, ocultando informações que, possivelmente, o usuário da classe não está interessado em saber (ou pelo menos não deveria estar);

Encapsulamento

- O uso de modificadores de acesso, juntamente com *getters* e *setters*, nos permite aplicar o conceito de **encapsulamento**;
- **Encapsulamento**: fornecer interfaces "seguras" de acesso aos componentes de sua classe, ocultando informações que, possivelmente, o usuário da classe não está interessado em saber (ou pelo menos não deveria estar);
- Outro exemplo: classe *Time2*.

Outros Conceitos

- Composição;
- Atributos *static*.

Exercícios

1. Explique por que uma classe pode e deve fornecer um método *set* e um método *get* para atributos de uma classe;

Exercícios

2. Crie uma classe *GradeBook* com as seguintes características:
   - Atributos privados *courseName* (*String*) e *instructorName* (*String*);
   - Construtor que recebe apenas um *courseName* como parâmetro. O atributo *instructorName* deve receber "Not defined yet";
   - Construtor que recebe um *courseName* e um *instructorName* como parâmetro;
   - Métodos *setters* e *getters* para os atributos privados, com comportamento padrão;
   - Método *displayMessage*, que exibe uma mensagem de boas-vindas, apresentando o nome do curso e o nome do professor (se houver).

   Crie uma classe *GradeBookTest* que testa todos os atributos e métodos da classe *GradeBook*.

Exercícios

3. Crie uma classe *Account* com as seguintes características:
   - Atributo privado *balance* (*double*);
   - Construtor que recebe um *initialBalance* como parâmetro e o atribui para o atributo *balance*. O parâmetro *initialBalance* só deve ser atributo para *balance* se for maior que 0;
   - Método *credit*, que adiciona uma determinada quantia à conta. A quantia deve ser passada através de um argumento *amount* (*double*);
   - Método *getter* para a variável *balance*;
   - Método *debit*, que retira dinheiro de uma *Account*. Assegure que a quantidade de débito não exceda o saldo de *Account*. Se exceder, o saldo deve ser deixado inalterado e o método deve imprimir uma mensagem que indica "Debit amount exceeded account balance".

   Crie uma classe *AccountTest* que testa todos os atributos e métodos da classe *Account*.

Exercícios

4. Crie uma classe chamada *Invoice* para que uma loja de suprimentos de informática possa utilizá-la para representar uma fatura de um item vendido na loja. Uma *Invoice* deve incluir quatro informações como atributos: número (*String*), descrição (*String*), quantidade comprada de um item (*int*) e o preço por item (*double*). Sua classe deve ter um construtor que inicializa os quatro atributos. Todos os atributos são privados e você deve fornecer *setters* e *getters*. Forneça também um método *getInvoiceAmount*, que calcula o total da fatura (isto é, multiplica a quantidade pelo preço por item) e retorna um *double*. Se a quantidade comprada não for positiva, ela deve ser configurada como 0. Se o preço por item não for positivo, ele deve ser configurado como 0.0. Escreva um aplicativo teste chamado *InvoiceTest* que demonstra as capacidades da classe *Invoice*.

Exercícios

5. Crie uma classe chamada *Employee* que inclua três atributos: primeiro nome (*String*), sobrenome (*String*) e salário mensal (*double*). Forneça um construtor que inicializa os três atributos. Forneça um método *set* e um *get* para cada atributo. Se o salário mensal não for positivo, não configure seu valor. Escreva uma classe de teste chamada *EmployeeTest* que demonstra as capacidades da classe *Employee*. Crie dois objetos *Employee* e exiba o salário anual de cada objeto. Então dê a cada *Employee* um aumento de 10% e exiba novamente o salário anual de cada *Employee*.

Exercícios

6. Crie uma classe chamada *Date* que inclua três atributos: dia (*int*), mês (*int*) e ano (*int*). Forneça um construtor que inicializa os três atributos e suponha que os valores fornecidos estejam corretos. Forneça um método *set* e um *get* para cada variável de instância. Forneça um método *displayDate* que exibe o dia, o mês e o ano separados por barras normais (/). Escreva uma classe de teste chamada *DateTest* que demonstra as capacidades da classe *Date*.

Exercícios

7. Modifique a classe Date criada anteriormente, para que ela realize uma verificação de erros nos valores inicializadores dos atributos *month*, *day* e *year*. Forneça um método *nextDay* para incrementar o dia por um. O objeto *Date* deve permanecer em um estado consistente. Escreva um programa que testa o método *nextDay* em um loop que imprime a data durante cada iteração para ilustrar que esse método funciona corretamente. Teste os seguintes casos:
   - Incrementar para o próximo mês;
   - Incrementar para o próximo ano.

Exercícios

8. Ao realizar exercícios físicos, você pode utilizar um monitor de frequência cardíaca para ver se sua frequência permanece dentro de um intervalo seguro, sugerido pelos seus treinadores e médicos. Segundo a a *American Heart Association*, a fórmula para calcular a *frequência cardíaca máxima* por minuto é 220 menos a idade. Sua *frequência cardíaca alvo* é um intervalo entre 50%-85% da frequência cardíaca máxima. Crie uma classe chamada *HeartRates*. Os atributos da classe devem incluir o nome, sobrenome e data de nascimento da pessoa (utilize a classe *Date* criada anteriormente). Sua classe deve ter um construtor que recebe esses dados como parâmetros. Para cada atributo, forneça métodos *set* e *get*. A classe também deve incluir um método que calcula e retorna a idade da pessoa (em anos), um método que calcula e retorna a frequência cardíaca máxima da pessoa e um método que calcula e retorna a frequência cardíaca-alvo da pessoa. Escreva uma classe de teste *HeartRatesTest* que solicita as informações da pessoa, instancia um objeto da classe *HeartRates* e logo após imprime as seguintes informações a partir desse objeto: nome, sobrenome, data de nascimento, idade, intervalo de frequência cardíaca máxima e frequência cardíaca-alvo.

Exercícios

9. Escreva uma classe *HealthProfile* que representará o perfil da saúde de uma pessoa cadastrada em um sistema. Sua classe deve conter os atributos nome *String*, sobrenome *String*, sexo *char*, data de nascimento (do tipo *Date* criado anteriormente), altura (*float*, em metros) e peso (*float*, em quilogramas). Sua classe deve ter um construtor que recebe esses dados. Para cada atributo, forneça métodos *set* e *get*. A classe também deve incluir métodos que calculem e retornem a idade do usuário em anos, intervalo de frequência cardíaca máxima, frequência cardíaca-alvo e índice de massa corporal (IMC). Escreva uma classe *HealthProfileTest* que solicite informações da pessoa, instancie um objeto da classe *HealthProfile* para essa pessoa e imprime as informações a partir desse objeto.

Exercícios

10. Crie uma classe *Rectangle* com atributos *length* e *width*, cada um dos quais assume o padrão de *1*. Forneça métodos que calculem o perímetro e a área do retângulo. A classe tem métodos *set* e *get* para o comprimento (*length*) e a largura (*width*). Os métodos *set* devem verificar se *length* e *width* são, cada um, números de ponto flutuante maiores que $0,0$ e menores que $20,0$. Escreva uma classe *RectangleTest* para testar a classe *Rectangle*.

Exercícios

11. Crie uma classe *SavingsAccount*. Utilize uma variável *static annualInterestRate* para armazenar a taxa de juros anual para todos os correntistas. Cada objeto da classe contém um atributo privado *savingsBalance* para indicar a quantidade que o cliente atualmente tem em depósito. Forneça método *calculateMonthlyInterest* para calcular os juros mensais multiplicando o *savingsBalance* por *annualInterestRate* dividido por 12 - esses juros devem ser adicionados ao *savingsBalance* Forneça um método *static modifyInterestRate* que configure *annualInterestRate* como um novo valor. Escreva um programa para testar a classe *SavingsAccount*. Instancie dois objetos *savingsAccount*, *saver1* e *saver2*, com saldos de R$ 2.000,00 e R$ 3.000,00, respectivamente. Configure *annualInterestRate* como 4% e então calcule o juro mensal de cada um dos 12 meses e imprima os novos saldos para os dois poupadores. Em seguida, configure *annualInterestRate* para 5%, calcule a taxa do próximo mês e imprima os novos saldos para os dois poupadores.

Exercícios

12. Crie uma classe chamada *Complex* para realizar aritmética com números complexos. Os números complexos têm a forma

    parteReal + *parteImaginária*i*

    onde *(I)* é $\sqrt{1}$.

    Escreva um programa para testar sua classe. Utilize variáveis de ponto flutuantes para representar os dados *private* da classe. forneça um construtor que permita que um objeto dessa classe seja inicializado quando ele for declarado. Forneça um construtor sem argumento com valores padrão caso nenhum inicializados seja fornecido.

Exercícios

12. Formeça métodos *public* que realizam as seguintes operações:

    - Somar dois números *Complex*: as partes reais são somadas de um lado e as partes imaginárias são somadas de outro;
    - Subtrair dois números *Complex*: a parte real do operando direito é subtraída da parte real do operando esquerdo e a parte imaginária do operando direito é subtraída da parte imaginária do operando esquerdo;
    - Imprimir números *Complex* na forma (a, b), onde a é a parte real e b é a parte imaginária.

Exercícios

13. Crie uma classe *Teclado* que contenha operações básicas de leitura do teclado. Decida quais serão as operações criadas, os modificadores de acesso dos métodos, se eles serão *static* e qual das classes Java você utilizará para leitura (*Scanner* ou *JOptionPane*). Crie uma classe *KeyboardTest* que teste as funcionalidades da classe criada.