

Orientação a Objetos

Prof. Ms. Peter Jandl Junior

Prof. Ms. Télvio Orru

Prof. Nathan Silva

J12B

Linguagem de Programação Orientada a Objetos

Ciência da Computação - UNIP – Jundiaí

POO :: conceitos básicos e aplicação

- Esta apresentação contém a definição dos conceitos básicos da Programação Orientada a Objetos – POO – e também sua aplicação na construção de classes por meio da linguagem de programação Java.
- A POO é um novo paradigma de programação de computadores que requer o desenvolvimento de habilidades de modelagem de objetos.

Definição de Classes

Instanciação

Classes que usam classes

Aplicação

POO:Definição de classes

Programação Orientada a Objetos

- Novo paradigma de programação com relação a interpretação dos problemas.
- Exige:
 - *Observar* os objetos que existem no mundo, como eles são;
 - *Identificar* suas características e funcionalidades;
 - *Classificar*, ou seja, organizar tipos em função de suas semelhanças;
 - *Modelar* classes e criar objetos.



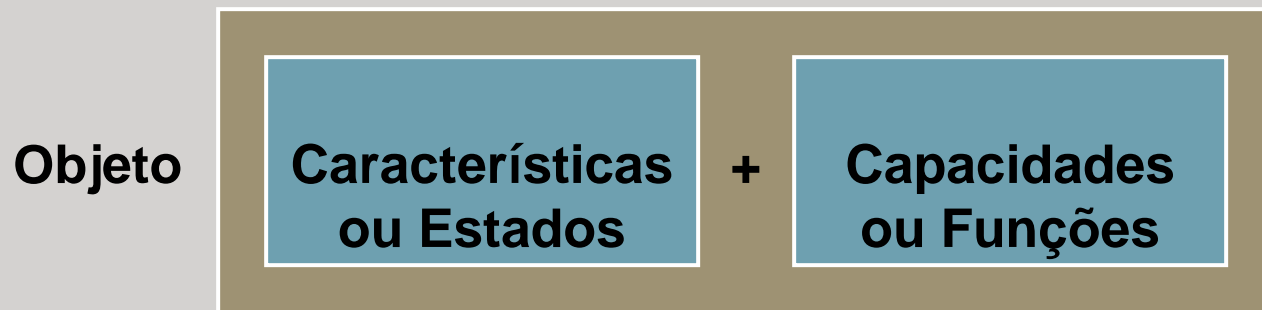
Modelagem de classes



- **Modelar** significa criar uma representação.
- Todo modelo é uma abstração, ou seja, deve incluir apenas os elementos necessários a solução de um problema.

Orientação a Objetos

- ❑ Modelo de programação mais natural do que a programação estruturada ou funcional.
- ❑ Entende tudo como objetos que pertencem a classes específicas.
- ❑ Classes agrupam as características (atributos) e capacidades (métodos) comuns dos objetos.



Princípios da OO

- Uma **classe** (= tipo) é um modelo, um conceito **abstrato** para famílias de objetos. A classe contém as definições do objeto.
- Um **objeto** é um representante real, uma entidade **concreta** de uma classe e pode ser distinguido dos demais objetos de sua classe e de outras. Um objeto contém os valores e o estado que representam uma entidade.

POO: modelando uma solução

Problema

Sábado	● Nova
2	
Domingo	
3	
Segunda	
4	
Terça	<i>Carnaval</i>
5	
Quarta	<i>Cinzas</i>
6	
Quinta	
7	
Sexta	
8	
Sábado	● Crescente
9	
Domingo	
10	

- Como representar uma lista de eventos (ou compromissos) em termos de sua:
 - Descrição
 - Data
 - Horário
- Com uma lista de eventos é possível organizar uma agenda, aplicativo de lembretes ou cronograma específico.

Modelagem de classe::Horario

- **Atributos** [características]:
 - horas
inteiro
0 .. 23
 - minutos
0 .. 59
- **Operações** [capacidades]:
ajuste dos atributos



Estrutura de uma classe Java

- A classe *pode* possuir diversos **campos** (atributos ou variáveis-membro) para representar seus estados.
- A classe *pode* possuir vários **métodos**, ou seja, funções-membro sobre seus estados.
- A classe também *pode* possuir diversos **construtores** destinados a realizar as operações de construção (criação) de seus objetos.

Estrutura de uma classe Java

```
// nível de acesso e nome
acesso class NomeDaClasse {
    // declaração dos campos da classe
    // acesso, tipo e nome
    acesso tipo nomeDoCampo

    // declaração dos métodos da classe
    // acesso, retorno, nome e lista de parâmetros
    acesso tipo nomeDoMétodo (lista_parâmetros) {
    }


    // declaração dos construtores da classe
    // acesso nome (= da classe) e lista de parâmetros
    acesso nomeDaClasse (lista_parâmetros) {
    }
}
```

Horario::atributos

- Representação de **horas**:
Tipo: int,
 com restrição de valor [0..23]
- Declaração do atributo (variável-membro):

 `public int horas;`

Se for *public*, não tem como garantir a restrição de valores necessária.

 `private int horas;`

Garante que restrição de valores seja aplicada, mas como acessar?

Horario::métodos set e get

- Métodos típicos (mas não obrigatórios) para realizar o acesso a campos de uma classe:
 - Método *set* efetua o ajuste do valor do campo (e portanto permite aplicar a regra de validação).
 - Método *get* retorna o valor do campo (garantindo privacidade).

```
public void setHoras (int h) {  
    if (h<0 || h>23) {  
        throw new RuntimeException("hora invalida: " + h);  
    }  
    horas = h;  
}  
  
public int getHoras () {  
    return horas;  
}
```

Se o valor do atributo não mudar, o método set deve ser *protected* ou *private*.

Horario::atributos

- Representação de **minutos**:
Tipo: int,
 com restrição de valor [0..59]
- Declaração do atributo (variável-membro):

 `public int minutos;`

Se for *public*, não tem como garantir a restrição de valores necessária.

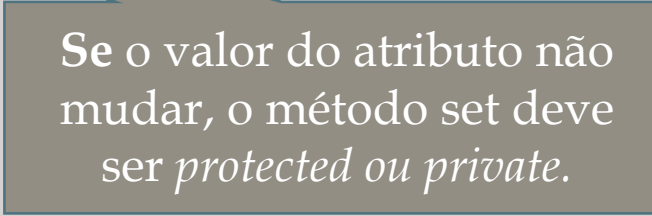
 `private int minutos;`

Garante que restrição de valores seja aplicada, mas como acessar?

Horario::métodos set e get

- Métodos típicos (mas não obrigatórios) para realizar o acesso a campos de uma classe:
 - Método *set* efetua o ajuste do valor do campo (e portanto permite aplicar a regra de validação).
 - Método *get* retorna o valor do campo (garantindo privacidade).

```
public void setMinutos (int m) {  
    if (m<0 || m>59) {  
        throw new RuntimeException("minuto invalido: " + m);  
    }  
    minutos = m;  
}  
  
public int getMinutos () {  
    return minutos;  
}
```



Se o valor do atributo não mudar, o método set deve ser *protected* ou *private*.

Horario::construtor

- Ao criar um horário:
 - Existe um horário-padrão (valor *default*)?
 - Faz sentido criar um novo horário sem definir seus valores de horas e minutos?

// construtor

```
public Horario (int h, int m) {  
    setHoras(h);  
    setMinutos(m);  
}
```

Instanciação::criação de novos objetos

- Declaração do objeto:
Horario h;
- Instanciação do objeto:
// construtor requer 2 valores:
// valor das horas e dos minutos do horário
h = new Horario (____ , ____);

Valor das *horas*:
um número inteiro
entre 0 e 23.

Valor dos *minutos*:
um número inteiro
entre 0 e 59.

Instanciação::criação de novos objetos

- `Horario h;`
- `h = new Horario(12, 30);`
ou
- `h = new Horario(19, 10);`
ou
- `h = new Horario(4, 15);`
ou
- `h = new Horario(0, 0);`

Instanciação::criação de novos objetos

- Horario h;
- **h = new Horario(17, 75);**
uso de valores inválidos provoca exceção:

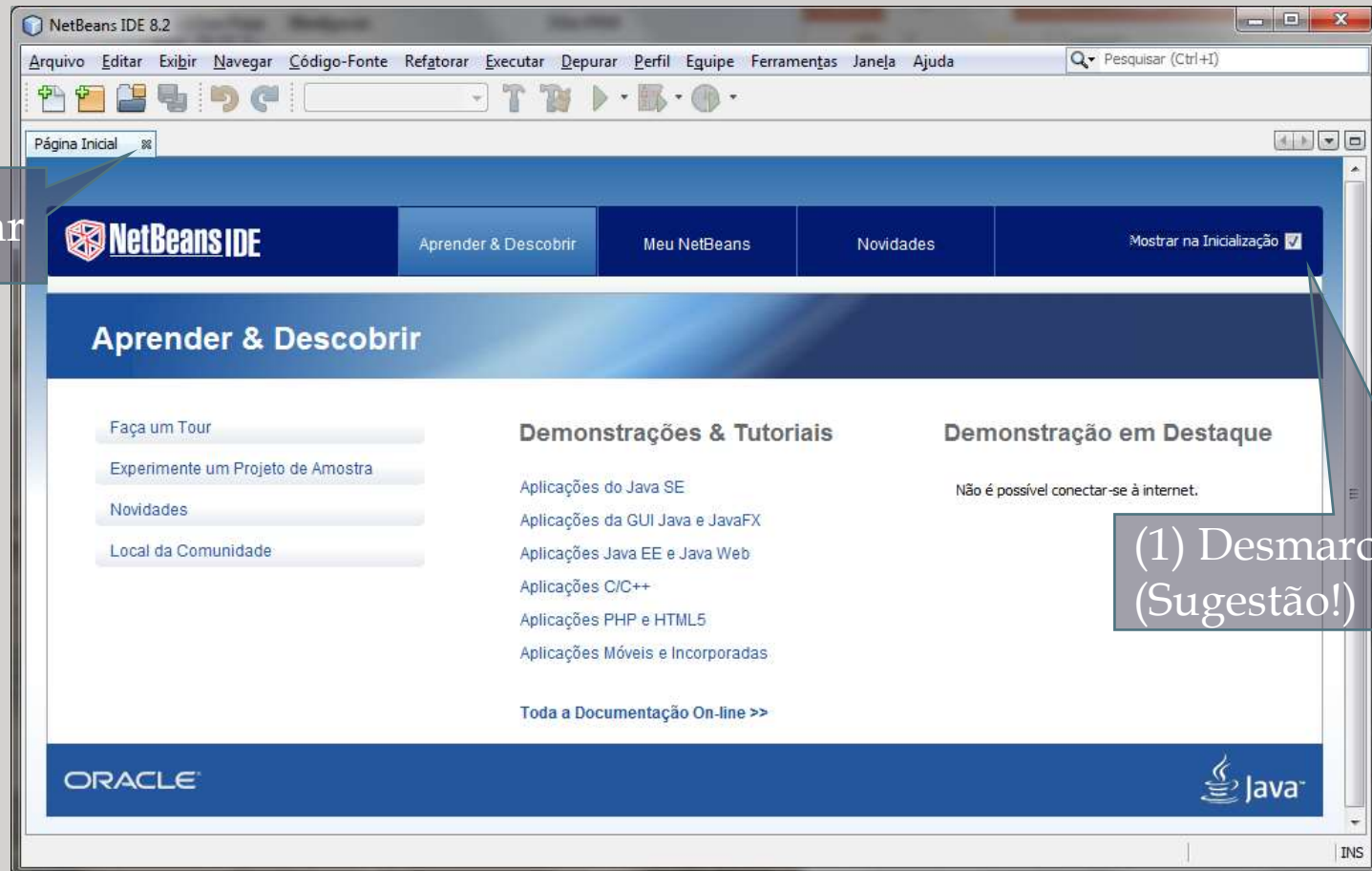
Exception in thread "main"
java.lang.RuntimeException: minuto invalido: 75

at ...

NetBeans IDE

Primeiros Passos

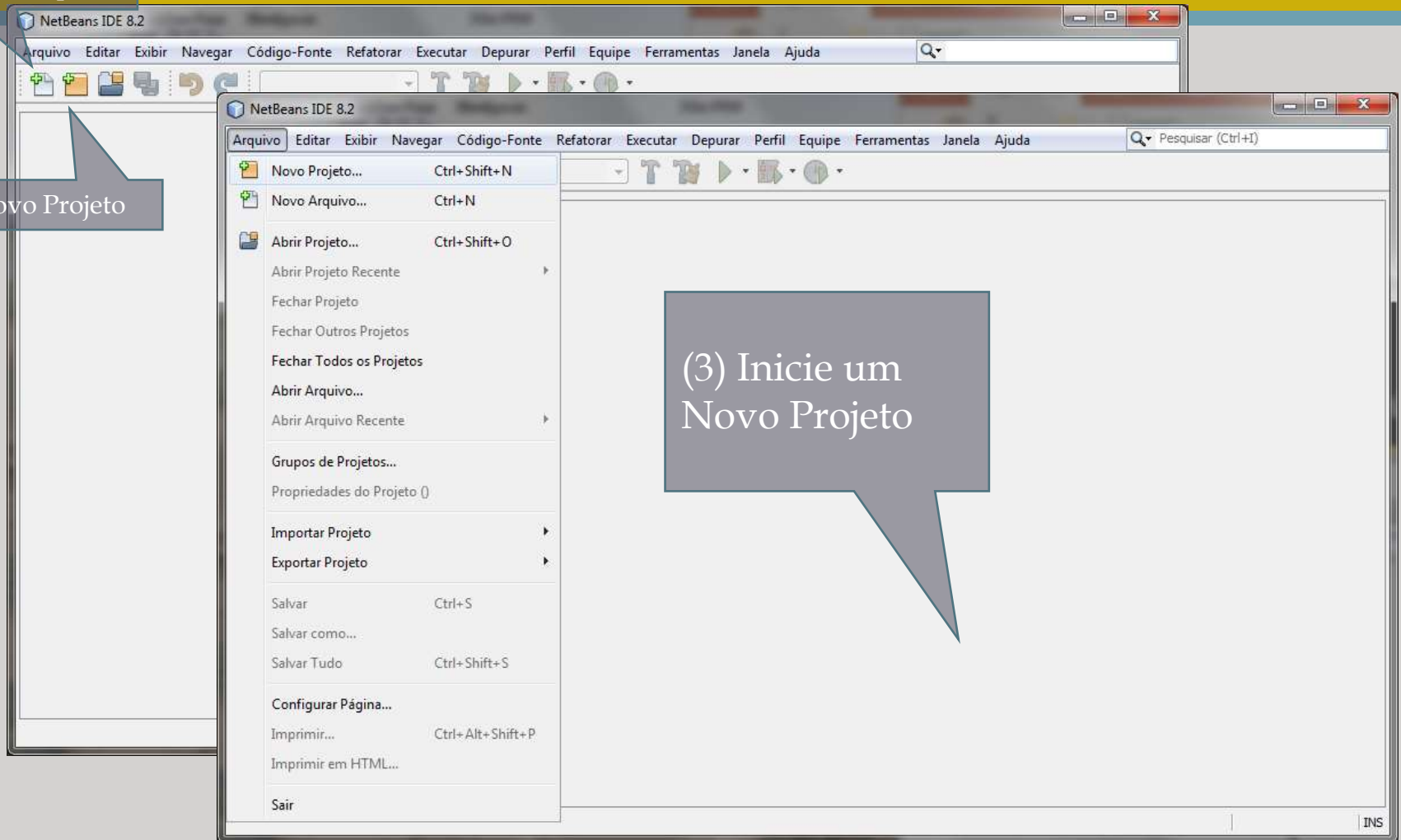
Acione o NetBeans por meio de seu ícone.



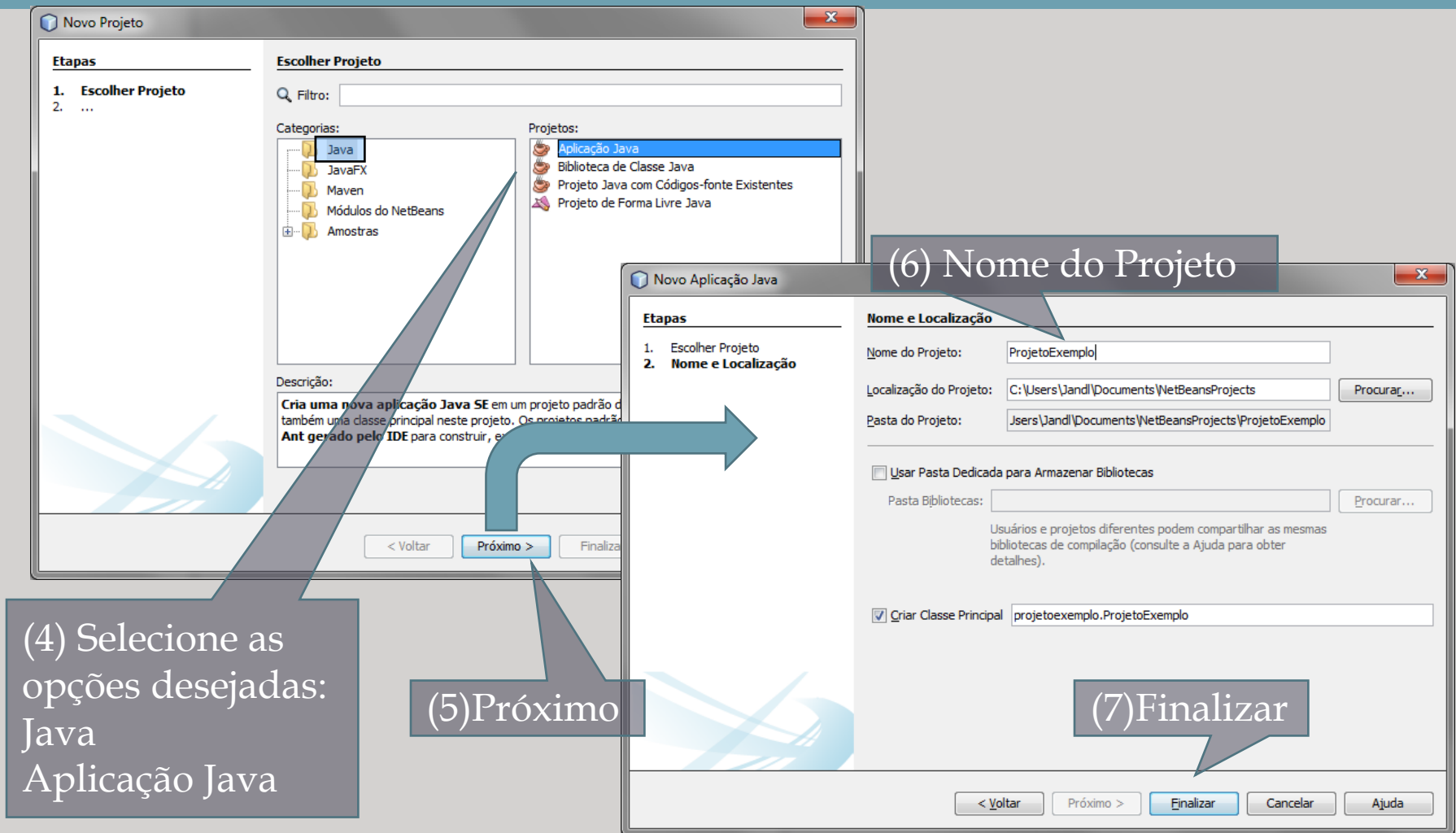
Para cada projeto ou exercício...

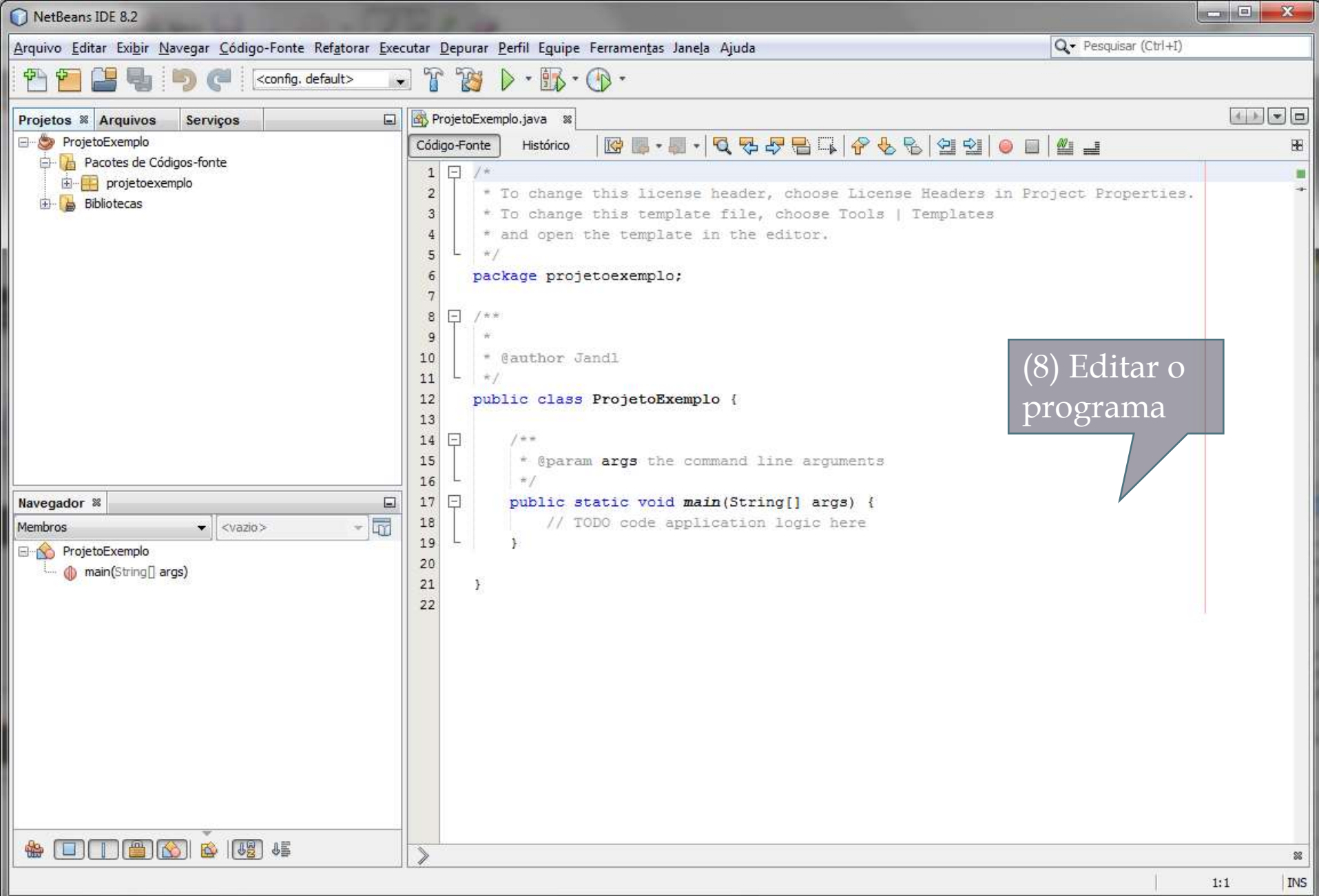
Novo Arquivo

Novo Projeto



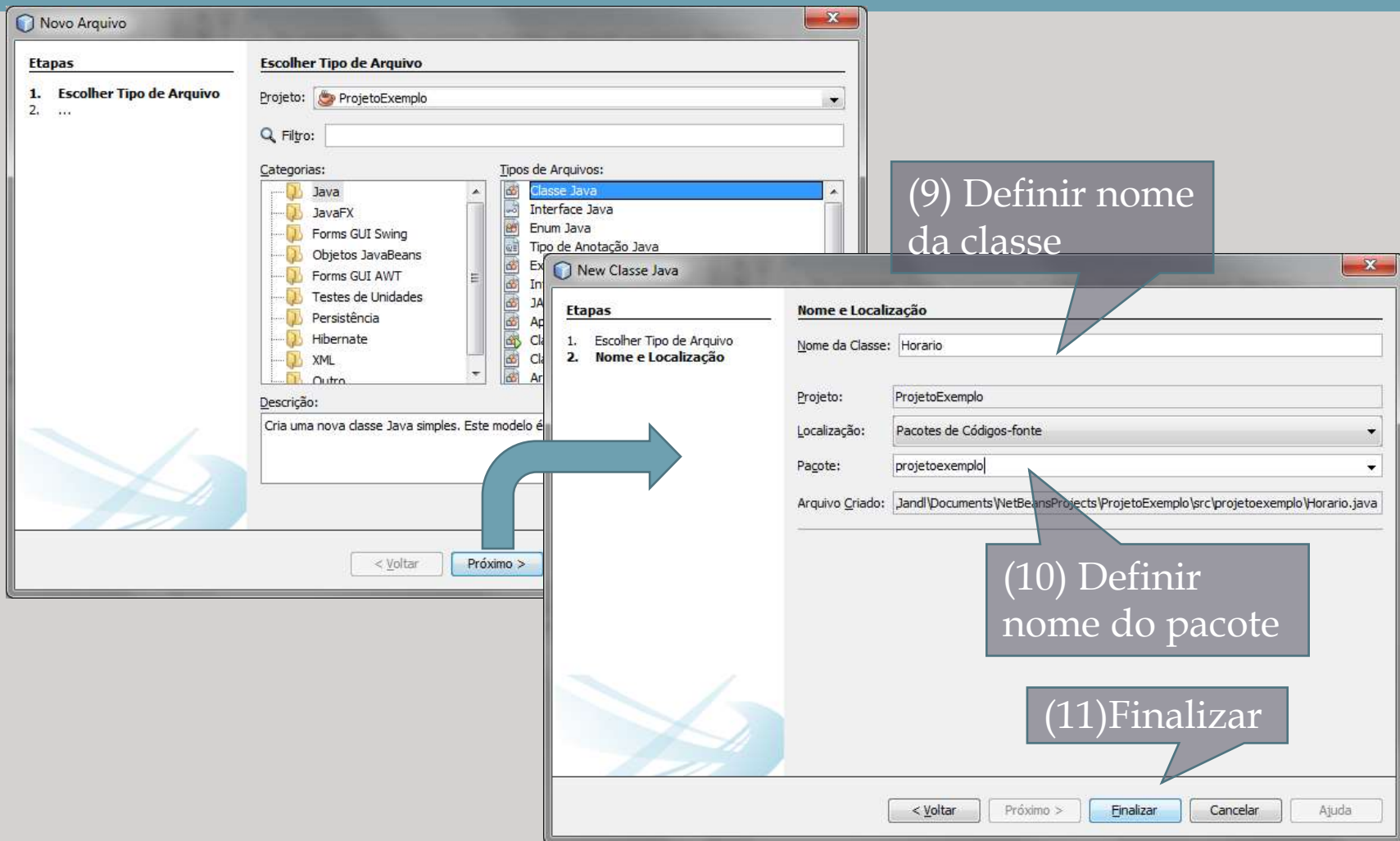
Categoria: *Java* | Projetos: *Aplicação Java* Nome do Projeto: *ProjetoExemplo*



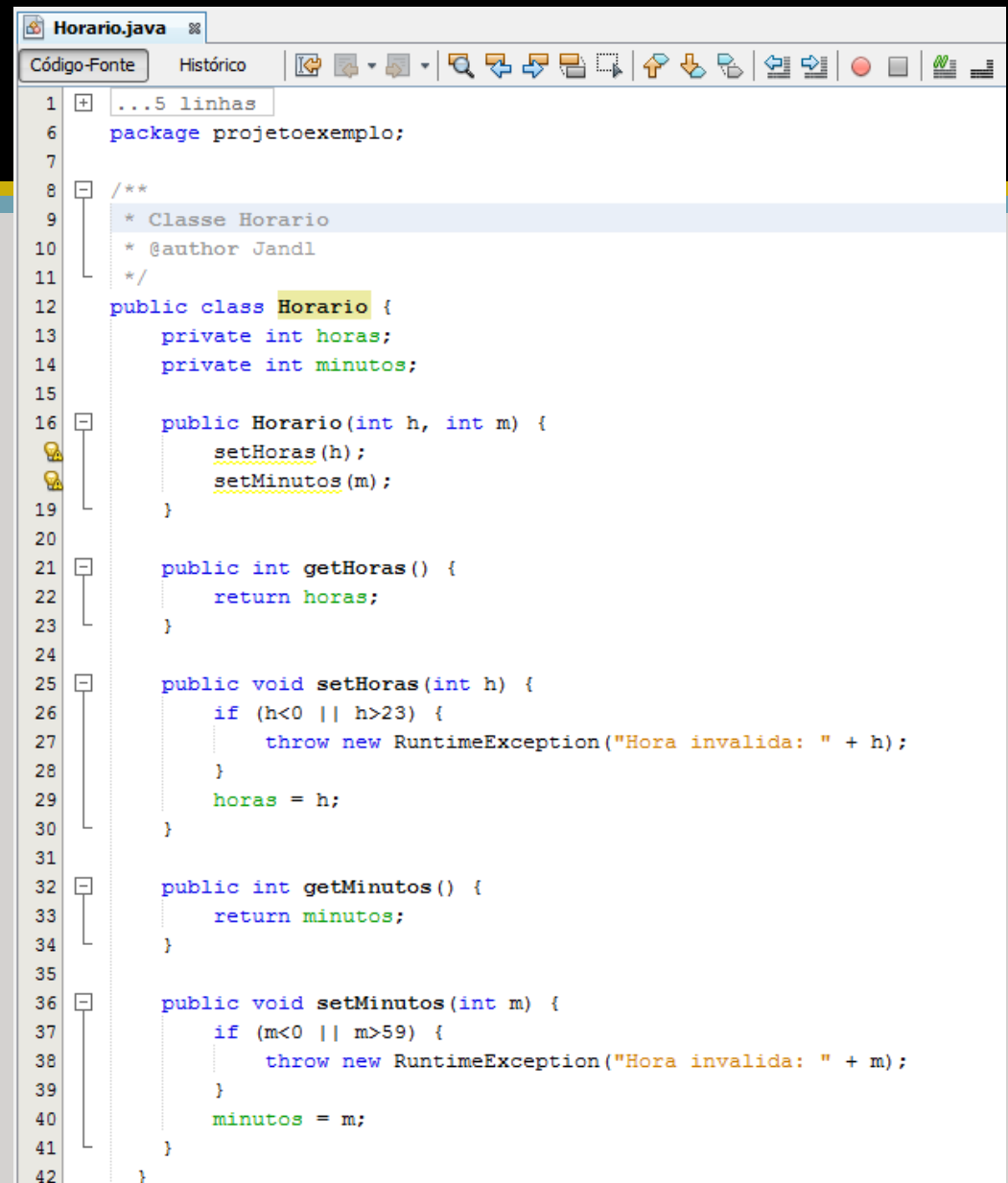


Novo Arquivo

Categoria: *Java* | Tipo: *Classe Java*

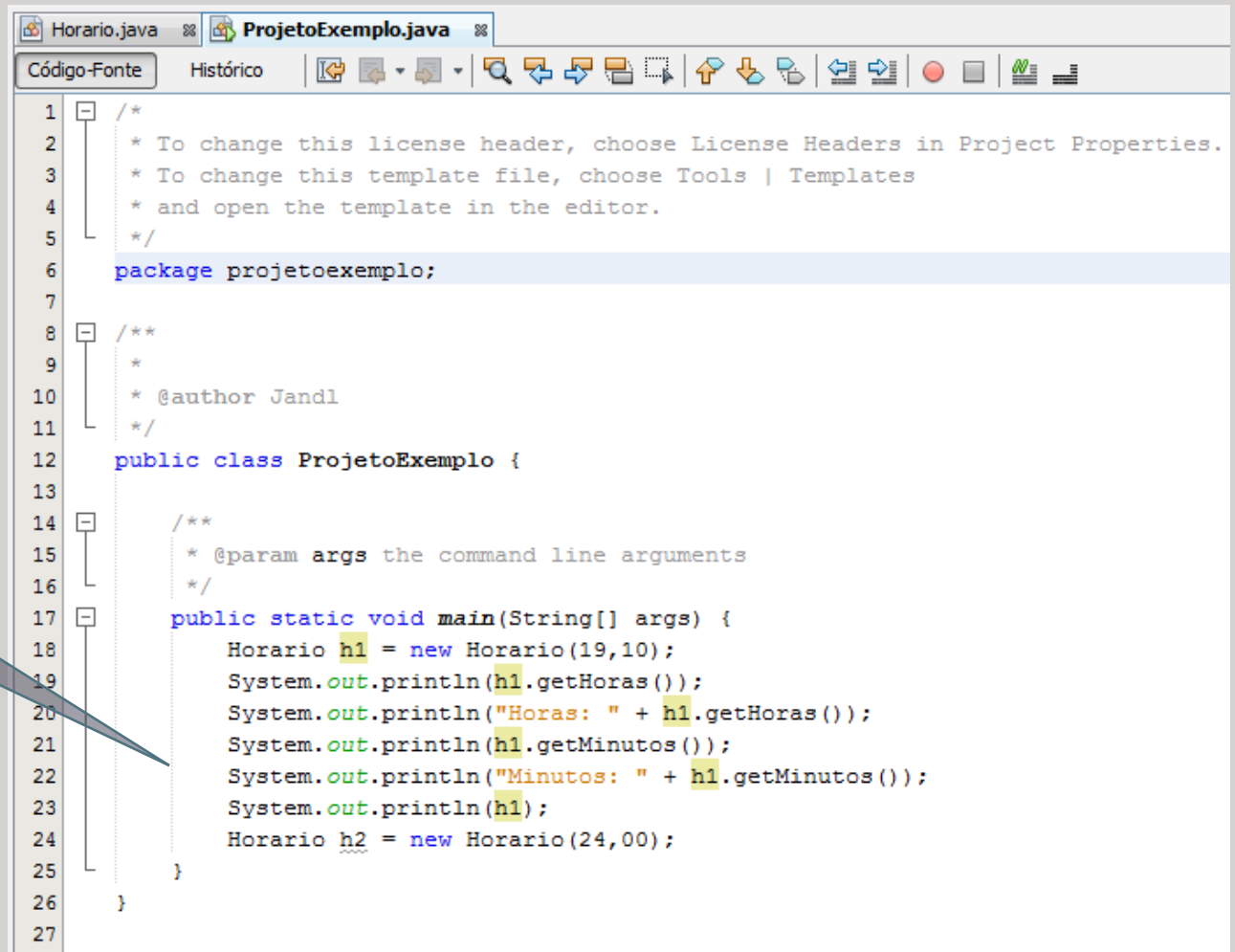


Classe Horario



```
Horario.java
Código-Fonte  Histórico
...5 linhas
1 package projetoexemplo;
2
3 /**
4  * Classe Horario
5  * @author Jandl
6  */
7 public class Horario {
8     private int horas;
9     private int minutos;
10
11     public Horario(int h, int m) {
12         setHoras(h);
13         setMinutos(m);
14     }
15
16     public int getHoras() {
17         return horas;
18     }
19
20     public void setHoras(int h) {
21         if (h<0 || h>23) {
22             throw new RuntimeException("Hora invalida: " + h);
23         }
24         horas = h;
25     }
26
27     public int getMinutos() {
28         return minutos;
29     }
30
31     public void setMinutos(int m) {
32         if (m<0 || m>59) {
33             throw new RuntimeException("Hora invalida: " + m);
34         }
35         minutos = m;
36     }
37 }
38
39
40
41
42
```

Um programa que usa Horario

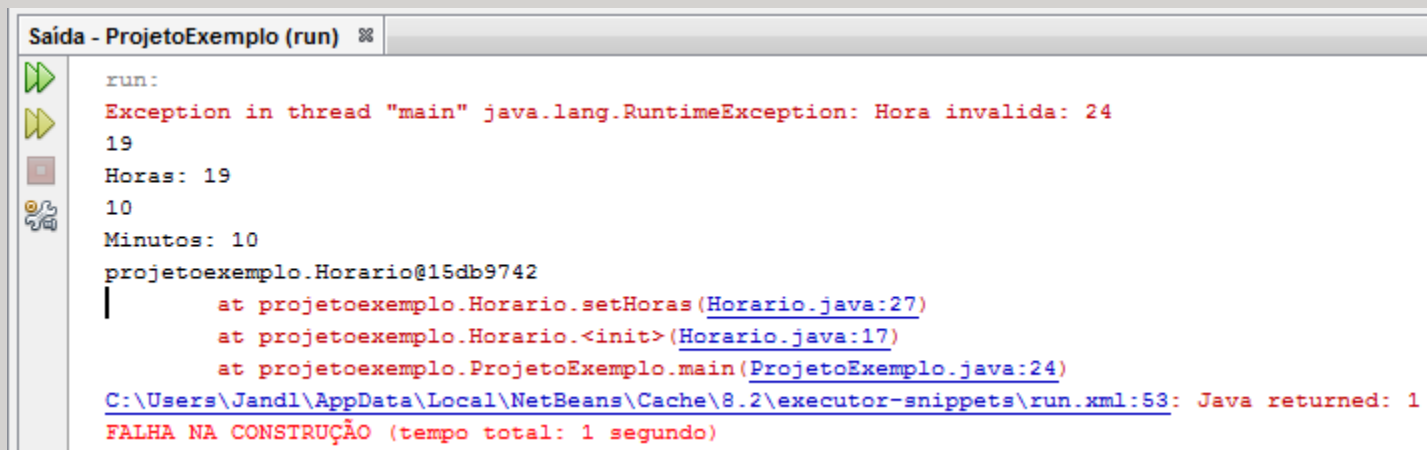


```
1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package projetoexemplo;
7
8  /**
9   *
10   * @author Jandl
11   */
12  public class ProjetoExemplo {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          Horario h1 = new Horario(19,10);
19          System.out.println(h1.getHoras());
20          System.out.println("Horas: " + h1.getHoras());
21          System.out.println(h1.getMinutos());
22          System.out.println("Minutos: " + h1.getMinutos());
23          System.out.println(h1);
24          Horario h2 = new Horario(24,00);
25      }
26  }
27
```

Edite o
programa
ProjetoExemplo

Execução do ProjetoExemplo

- Programa deve exibir dados do objeto **h1** corretamente.
- Mas deve exibir exceção e ser finalizado na instanciação do objeto **h2**.



```
Saída - ProjetoExemplo (run) ✖
run:
Exception in thread "main" java.lang.RuntimeException: Hora invalida: 24
19
Horas: 19
10
Minutos: 10
projetoexemplo.Horario@15db9742
|
    at projetoexemplo.Horario.setHoras (Horario.java:27)
    at projetoexemplo.Horario.<init> (Horario.java:17)
    at projetoexemplo.ProjetoExemplo.main (ProjetoExemplo.java:24)
C:\Users\Jandl\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
FALHA NA CONSTRUÇÃO (tempo total: 1 segundo)
```

Correções e Adições

- Acrescente o método abaixo ao final da classe **Horario**.

```
public String toString() {  
    return String.format("02d%:02d%", horas, minutos);  
}
```

- Corrija a instanciação do objeto **h2** em **ProgramaExemplo**.
- Complete o **ProgramaExemplo** para que sejam exibidos os dados (horas e minutos) de **h2**.
- *Execute e veja o efeito das alterações!*

POO:continuando a modelagem da solução

Modelagem de classe::Data

- **Atributos** [características]:
 - dia
inteiro
1 .. 31
 - mês
1 ..
 - ano
1900 .. 2100
- **Operações** [capacidades]:
ajuste dos atributos



Data::atributos

- Representação de **ano**:
Tipo: int,
 c/ restrição de valor [1900..2100]
- Declaração do atributo (variável-membro):

 `public int ano;`

Se for *public*, não tem como garantir a restrição de valores necessária.

 `private int ano;`

Garante que restrição de valores seja aplicada, mas como acessar?

Data::métodos set e get

- Métodos típicos (mas não obrigatórios) para realizar o acesso a campos de uma classe:
 - Método *set* efetua o ajuste do valor do campo (e portanto permite aplicar a regra de validação).
 - Método *get* retorna o valor do campo (garantindo privacidade).

```
public void setAno (int a) {  
    if (a<1900 || a>2100) {  
        throw new RuntimeException("ano invalido: " + a);  
    }  
    ano = a;  
}  
  
public int getAno () {  
    return ano;  
}
```

Data::atributos

- Representação de **mes**:
Tipo: int,
com restrição de valor [1..12]
- Declaração do atributo (variável-membro):

 `public int mes;`

Se for *public*, não tem como garantir a restrição de valores necessária.

 `private int mes;`

Garante que restrição de valores seja aplicada, mas como acessar?

Data::métodos set e get

- Métodos típicos (mas não obrigatórios) para realizar o acesso a campos de uma classe:
 - Método *set* efetua o ajuste do valor do campo (e portanto permite aplicar a regra de validação).
 - Método *get* retorna o valor do campo (garantindo privacidade).

```
public void setMes (int m) {  
    if (m<1 || m>12) {  
        throw new RuntimeException("mes invalido: " + m);  
    }  
    mes = m;  
}  
  
public int getMes () {  
    return mes;  
}
```

Data::atributos

- Representação de **dia**:
Tipo: int,
 com restrição de valor [1..31]
 com dependência do mês e ano
- Declaração do atributo (variável-membro):

 `public int dia;`

Se for *public*, não tem como garantir a restrição de valores necessária.

 `private int dia;`

Garante que restrição de valores seja aplicada, mas como acessar?

Data::métodos set e get

- Métodos típicos (mas não obrigatórios) para realizar o acesso a campos de uma classe:
 - Método *set* efetua o ajuste do valor do campo (e portanto permite aplicar a regra de validação).
 - Método *get* retorna o valor do campo (garantindo privacidade).

```
public void setDia (int d) {  
    int max = ???;  
    if (d<1 || d>max) {  
        throw new RuntimeException("dia invalido: " + d);  
    }  
    dia = d;  
}  
  
public int getDia () {  
    return dia;  
}
```

Um bom exercício de lógica! Sugestão: crie um método auxiliar!

Data::construtor

- Ao criar uma Data:
 - Existe uma data-padrão (valor *default*)?
 - Faz sentido criar uma data horário sem definir seus valores de dia, mês e ano?

// construtor

```
public Data (int d, int m, int a) {  
    setAno(a);  
    setMes(m);  
    setDia(d);  
}
```

Instanciação::criação de novos objetos

- Declaração do objeto:
Data d;
- Instanciação do objeto:
// construtor requer 3 valores:
// valor do dia, mês e ano da data
- `d = new Data (____ , ____ , ____);`

Valor do *dia*:
um número inteiro
entre 1 e 31, que
depende do mês.

Valor do *mês*:
um número inteiro
entre 1 e 12.

Valor do *ano*:
um número inteiro
entre 1900 e 2100.

Construa a classe Data no NetBeans

- No mesmo projeto...
- Adicione uma classe Data
- Edite o código como indicado nos slides
- Modifique o ProgramaExemplo para instanciar duas datas diferentes, exibindo seus valores!

Modelagem de classe::Evento

- **Atributos** [características]:
 - descrição
String
deve possuir algum conteúdo
 - data
qualquer que seja válida
 - horário
opcional (sem horário
- **Operações** [capacidades]:
ajuste dos atributos



Evento::atributos

- Representação de **descricao**:
Tipo: String,
com conteúdo
- Declaração do atributo (variável-membro):

 `public String descricao;`

 `private String descricao;`

Evento::métodos set e get

```
// descrição requer conteúdo, ou seja,  
// objeto não nulo, com conteúdo diferentes de brancos  
public void setDescricao (String d) {  
    if (d==null || d.trim().length()==0) {  
        throw new RuntimeException("descricao invalida");  
    }  
    descricao = d;  
}  
  
public String getDescricao () {  
    return descricao;  
}
```

Evento::atributos

- Representação de **data**:
Tipo: Data,
obrigatória e válida
- Declaração do atributo (variável-membro):

 `public Data data;`

 `private Data data;`

Evento::métodos set e get

```
// data é obrigatória, ou seja,  
// objeto não nulo, válido  
public void setData (Data d) {  
    if (d==null) {  
        throw new RuntimeException("data invalida");  
    }  
    data = d;  
}  
  
public Data getData () {  
    return data;  
}
```

Evento::atributos

- Representação de **horario**:
Tipo: Horario,
 opcional, válido
- Declaração do atributo (variável-membro):

 `public Horario horario;`

 `private Horario horario;`

Não seria incorreto,
neste caso, acesso
público.

Evento::métodos set e get

```
// horario é opcional, ou seja, quando existe,  
// requer objeto válido  
public void setHorario (Horario h) {  
    horario = h;  
}  
  
public Horario getHorario () {  
    return horario;  
}  
  
// método auxiliar, extra, que sinaliza evento de dia inteiro,  
// isto é, sem horário definido  
public boolean isDiaInteiro() {  
    return horário==null;  
}
```


Evento::construtor

- Ao criar um Evento:
 - Descrição é sempre requerida, pois não existe *evento-padrão*.
 - Faz sentido criar um Evento apenas com data (de dia inteiro).
 - Opcionalmente pode ser indicado o horário

// construtor

```
public Evento (String desc, Data data, Horario h) {  
    setDescricao(desc);  
    setData(data);  
    setHorario(h);  
}  
  
public Evento (String desc, Data data) {  
    this(desc, data, null); // desta forma acionamos o outro construtor  
}
```

Construa a classe Evento no NetBeans

- No mesmo projeto...
- Adicione uma classe Evento
- Edite o código como indicado nos slides

Novo Projeto

Novo Aplicação Java

Etapas

1. Escolher Projeto
2. **Nome e Localização**

Nome e Localização

Nome do Projeto:

Localização do Projeto:

Pasta do Projeto:

☐ Usar Pasta Dedicada para Armazenar Bibliotecas

Pasta Bibliotecas:

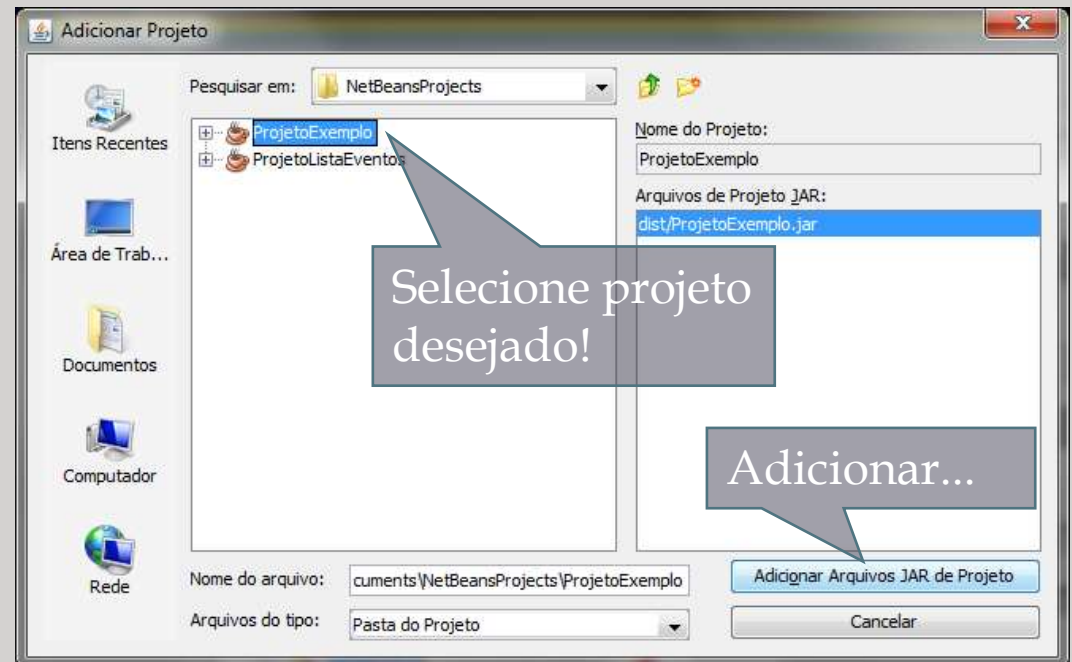
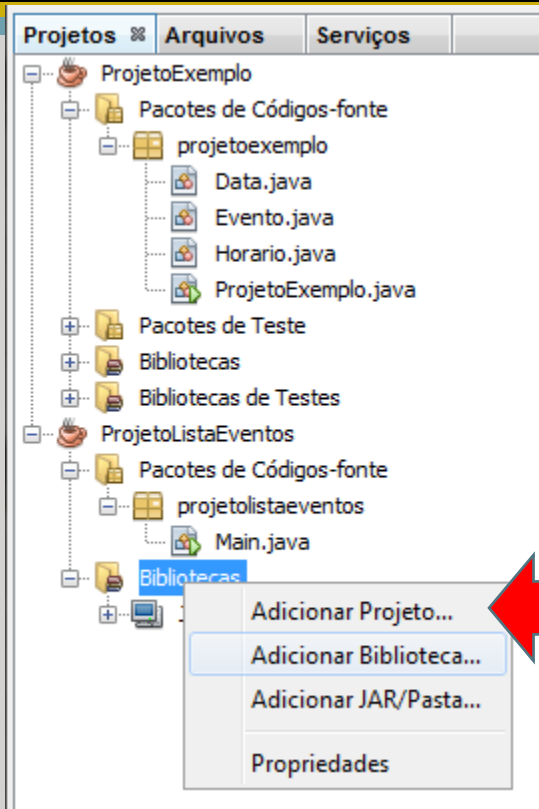
Usuários e projetos diferentes podem compartilhar as mesmas bibliotecas de compilação (consulte a Ajuda para obter detalhes).

☒ Criar Classe Principal

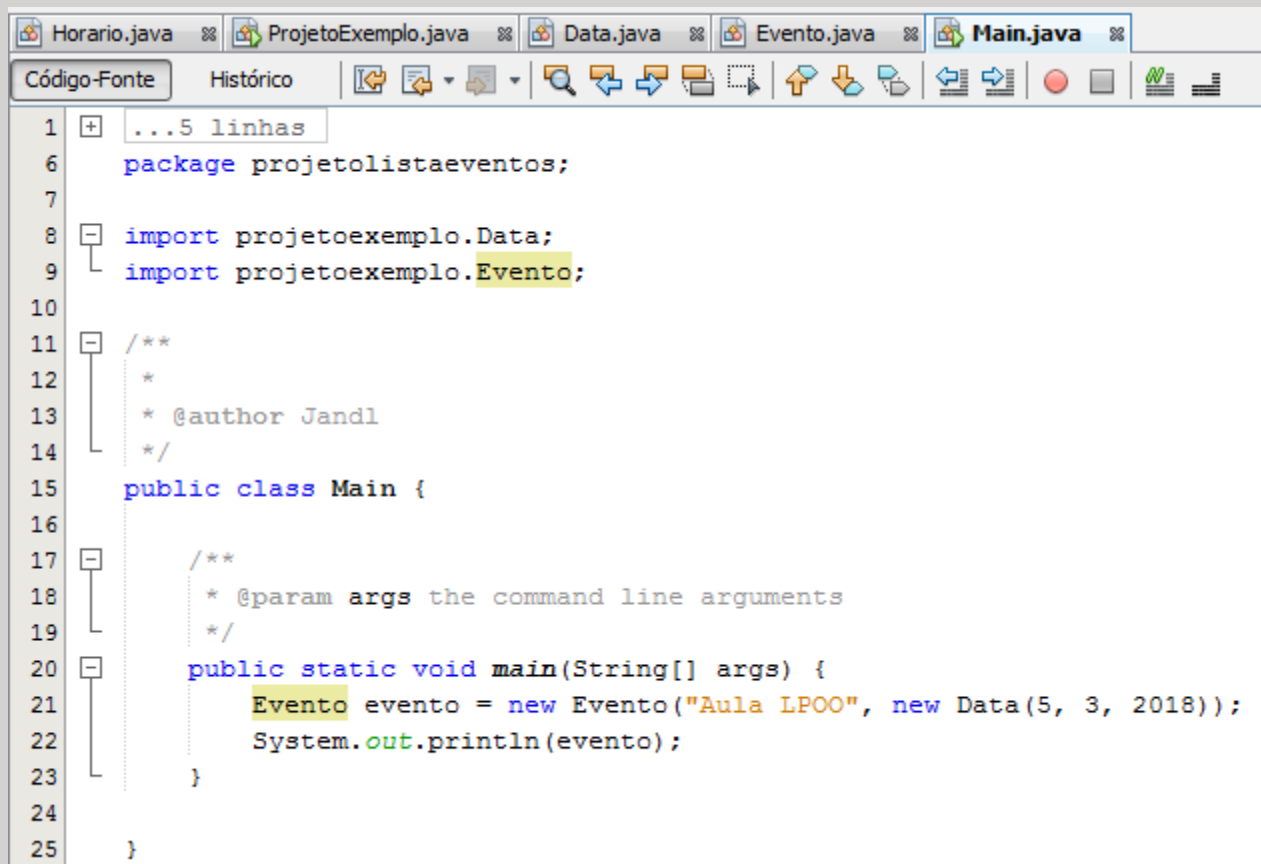
< Voltar Próximo > **Finalizar** Cancelar Ajuda

Adição de Projeto (como biblioteca)

- Selecione o novo projeto **ProjetoListaEventos**
- Acione botão direito em **Bibliotecas**
- Selecione **Adicionar Projeto**



Edite o código de Main



```
1  ...5 linhas
6  package projetolistaeventos;
7
8  import projetoexemplo.Data;
9  import projetoexemplo.Evento;
10
11  /**
12   *
13   * @author Jandl
14   */
15  public class Main {
16
17      /**
18       * @param args the command line arguments
19       */
20      public static void main(String[] args) {
21          Evento evento = new Evento("Aula LPOO", new Data(5, 3, 2018));
22          System.out.println(evento);
23      }
24
25  }
```

Recomendações de Estudo



- **Estudar com livros:**
 - Java – Guia do Programador, 3ª Ed, P. JANDL Jr, Novatec, 2015.
 - Java: Como Programar, 6ª Ed, DEITEL, H. & DEITEL, P., Pearson, 2007.