ORACLE

Academy

Java Foundations

7-5
Interação e Encapsulamento de Objetos

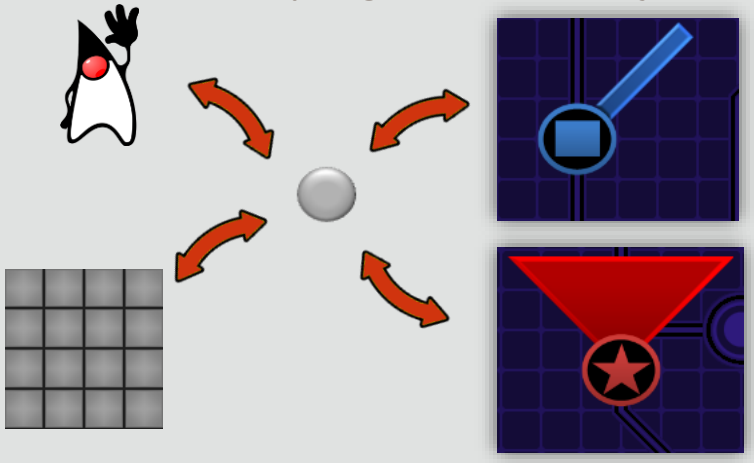
ORACLE
Academy

Objetivos

- Esta lição abrange os seguintes objetivos:
  - Entender a interação de objetos mais detalhadamente
  - Usar o modificador private para definir variáveis de classe
  - Entender a finalidade de métodos getter
  - Entender a finalidade de métodos setter

Interação de Objetos

- A Seção 2 mostrou a ideia de interação de objetos
  - Nenhuma sequência prescrita sobre como o objeto deve interagir
- Esta lição mostra como programar interações



O que É a Interação de Objetos?

- Uma referência a objeto é um endereço na memória
  - Uma referência direciona um objeto para outro
  - Uma referência permite que um objeto interaja com outro
- Os objetos interagem...
  - Acessando campos de outro objeto
  - Chamando métodos de outro objeto
- Se o método main instanciar cada objeto...
  - O método main contém uma referência a cada objeto
  - O método main pode acessar os campos e os métodos de cada objeto

Programa de Exemplo

- Considere um programa que modele os objetos Prisoner, Cell e Guard
- O método main teria esta aparência:

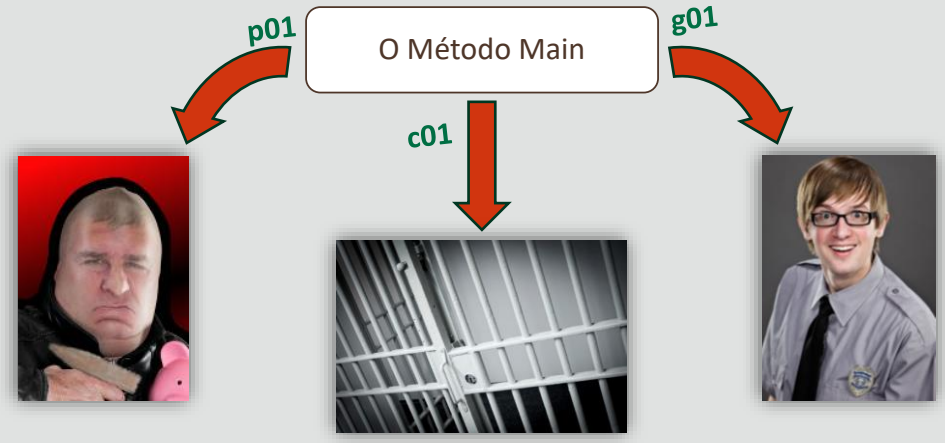
```
public class PrisonTest{
   public static void main(String[] args){
      Prison p01 = new Prisoner();
      Cell   c01 = new Cell();
      Guard  g01 = new Guard();

      p01.name = "Bubba";
      c01.name = "A1";
      g01.name = "Boss Man";
   }//fim do método main
}//fim da classe PrisonTest
```

Referências a objetos

Interações

Interações do Método Main

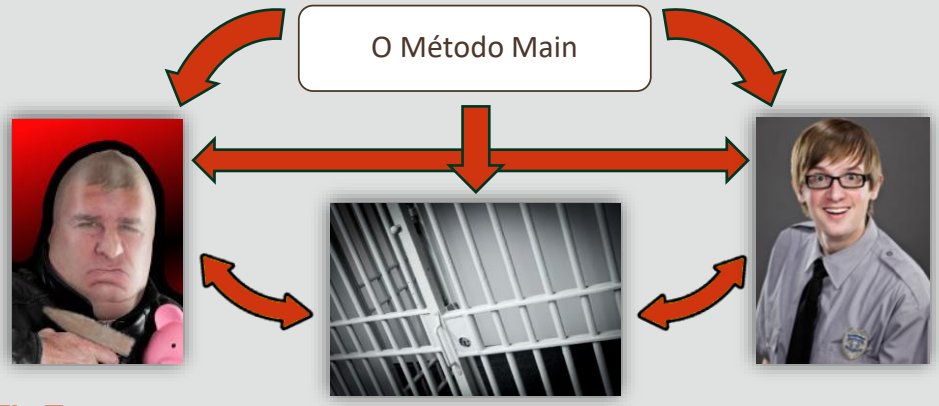
- O método main contém todas as referências a objetos
- Por isso, ele controla todas as interações nesse sistema



Observação para o Instrutor: as imagens nesses dois slides devem estar na mesma posição.

Interações entre Objetos

- No entanto, às vezes você desejará um programa em que os objetos interajam um com o outro
- Para isso, os objetos precisam saber um do outro
  - Um objeto precisa saber da referência ao outro objeto



Observação para o Instrutor: as imagens nesses dois slides devem estar na mesma posição.

Como os Objetos Sabem Uns sobre os Outros?

- As referências a objetos devem ser compartilhadas:
  - Um objeto pode conter um outro objeto como um campo
  - O método de um objeto pode aceitar um outro objeto como um argumento
- Por exemplo:
  - Uma maneira de descrever um Prisoner é pelo número da respectiva Cell
  - Poderia ser dito que uma Cell é uma propriedade de um objeto Prisoner
  - A classe Prisoner manteria um campo Cell

Exercício 1, Parte 1

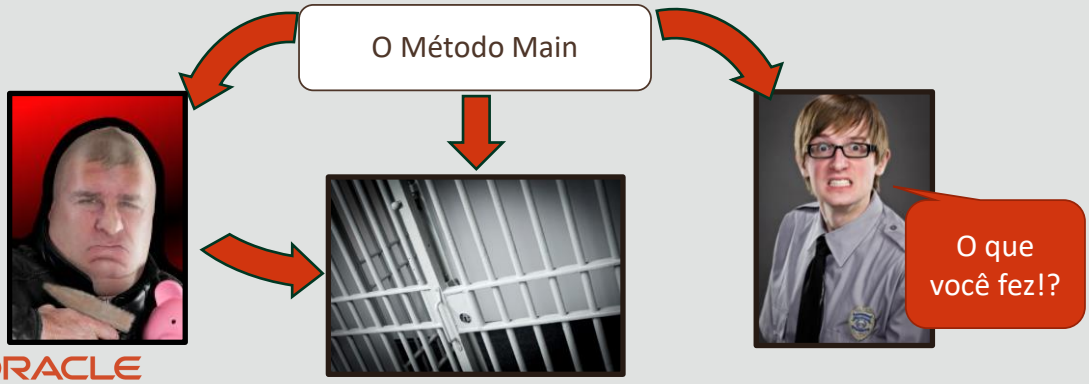
- Continue a editar o projeto `PrisonTest`
  - Uma versão deste programa é fornecida nos arquivos `PrisonTest_Student_7_5.java` e `Prisoner_Student_7_5.java`
- Crie uma classe Cell que inclua o seguinte:
  - A string name da cela
  - Um valor booliano que descreva se a porta está aberta
  - Um construtor com dois argumentos que defina os dois campos
- Modifique a classe Prisoner de modo que ela:
  - Inclua um campo Cell
  - Defina o campo Cell com base em um parâmetro do construtor
  - Imprima o name da cela como parte do método display()

Exercício 1, Parte 2

- Escreva um método openDoor()na classe Prisoner
  - Acesse e modifique o campo correspondente no objeto Cell de modo que:
    - Se a porta estiver fechada, abra-a
    - Se a porta estiver aberta, feche-a
  - Imprima se a porta abre ou fecha
- No método main:
  - Instancie uma Cell e um objeto Prisoner
  - Chame o método display() do prisioneiro uma vez
  - Chame o método openDoor() algumas vezes

Opa!

- Os guardas estão em pânico!
- Seu programa permite que os prisioneiros tenham acesso às portas das celas
- Considerando os planos de vingança de Bubba, esse tipo de interação não deve ser permitida!



Considere as Interações Possíveis entre os Objetos

- Considere quais objetos devem saber uns sobre os outros
  - Alguns objetos não têm por que modificar os campos de um outro objeto
  - Tente minimizar o conhecimento que os objetos têm uns dos outros...
  - Isso evita resultados indesejados e torna o código menos complicado
- Considere em qual direção as interações devem ocorrer e quais objetos devem ser propriedades uns dos outros
  - Um Prisoner deve ter uma propriedade Cell?
  - Uma Cell deve ter uma propriedade Prisoner?
  - Ou nenhum dos dois objetos deve ter conhecimento um do outro?

Considere como Distribuir Comportamentos

- As celas estão projetadas para abrirem e fecharem
  - Alguém precisa ter acesso para executar essas interações
  - Os prisioneiros não devem ser capazes desse comportamento
  - Os guardas não devem ser capazes desse comportamento
- É um desafio importante na programação orientada a objetos decidir como distribuir comportamentos entre os objetos
  - Mas não se preocupe. Você já tem experiência nisso.
  - Uma das principais metas do Java Puzzle Ball era criar situações em que os jogadores precisassem pensar com cuidado sobre como distribuir comportamentos entre tipos diferentes de objetos

Introdução ao Encapsulamento

- Às vezes, os objetos precisam saber uns sobre os outros
- O encapsulamento fornece técnicas para limitar a visibilidade de uma classe para outra
- É possível restringir quais campos e métodos outras classes podem ver
- Métodos especiais podem ser escritos para decidir como os dados devem ser acessados e modificados
- O acesso e a visibilidade devem ser limitados o máximo possível

Modificadores de Acesso

- A palavra-chave public é um dos vários modificadores de acesso
- Os modificadores de acesso limitam a visibilidade dos campos e métodos entre as classes

```
public class Cell {
   //Campos
   public String name;
   public boolean isOpen ;

   //Construtor
   public Cell(String name, boolean isOpen){
      this.name = name;
      this.isOpen = isOpen;
   }//fim construtor
}//fim da classe Cell
```

Detalhes sobre Modificadores de Acesso

- public: Visível para qualquer classe
  - É o menos seguro
  - Os métodos são tipicamente públicos
- Pacote: visível para o pacote atual
  - Não existe palavra-chave para este nível de acesso
- private: Visível somente para a classe atual
  - É o mais seguro
  - Em geral, os campos são privados

Exercício 2

- Continue a editar o projeto `PrisonTest`
- Modifique a classe `Cell`:
  - Altere seus campos para private
  - Salve o arquivo
- Seu IDE tem alguma reclamação?
  - Que reclamações?
  - Onde elas ocorrem?

Os Efeitos dos Dados Privados

- Os seguintes campos privados não podem ser acessados fora da classe Cell:
  - isOpen
  - name
- Nem mesmo o método main consegue acessar esses dados
- É bom que os prisioneiros não consigam abrir as portas de suas celas
- É ruim que os prisioneiros não saibam os nomes de suas celas
  - O próximo tópico analisa como solucionar esse problema

Introdução aos Métodos Getter

- Quando um campo está inacessível, ele não pode ser:
  - Lido
  - Modificado
- No entanto, em muitos casos, é desejável que uma classe pelo menos saiba o valor dos campos da outra classe
  - Um prisioneiro deve saber, pelo menos, o nome da sua cela
  - Isso requer que o prisioneiro leia o valor do campo name de uma Cell
- Os métodos getter fornecem uma solução

Métodos Getter

- Os getters também são denominados acessores
- Os getters são públicos
- Os getters normalmente não aceitam argumentos
- Os getters retornam o valor de uma variável específica
  - A maioria das variáveis privadas requer um método getter

```
public class Cell {
   …
   public String getName(){
      return name;
   }//fim do método getName
   public boolean getIsOpen(){
      return isOpen;
   }//fim do método getIsOpen
}//fim da classe Cell
```

Introdução aos Métodos Setter

- Em outros casos, é desejável que uma classe modifique o campo de outra classe
- No entanto, isso deve ser feito com segurança
- Por exemplo:
  - Um guarda deve ser capaz de abrir uma porta, mas um prisioneiro, não
  - Uma conta bancária não deve ter um valor inferior a zero
- Os métodos setter fornecem uma solução

Métodos Setter

- Os setters também são denominados modificadores
- Geralmente, eles são públicos
- Os setters normalmente aceitam argumentos
- Os setters são métodos do tipo void

```
public class Cell {
   …
   public void setName(String name){
      this.name = name;
   }//fim do método setName
   public void setIsOpen(boolean isOpen){
      this.isOpen = isOpen;
   }//fim do método setIsOpen
}//fim da classe Cell
```

Projetando Setters

- Tome cuidado ao escrever setters como esses mostrados no slide anterior
  - Os prisioneiros teriam novamente acesso às portas de suas celas
- Às vezes, é necessário fazer uma avaliação ao projetar um método setter
  - Uma porta de segurança pode solicitar um código de segurança
  - Um software bancário pode verificar se o valor de uma retirada resultaria em um saldo menor que zero ou se o valor retirado é negativo

Exercício 3, Parte 1

- Continue a editar o projeto `PrisonTest`
- Modifique a classe Cell para que…
  - Os getters existam para os campos name e isOpen
  - Existe um campo com um código de segurança com quatro dígitos Ele é inicializado no construtor e não tem um método getter
  - Existe um método setter para abrir/fechar a porta Ele faz o seguinte:
    - Aceita um código de segurança como argumento
    - Imprime se o código está incorreto
    - Se o código está correto e a porta está fechada, ele abre-a
    - Se o código está correto e a porta está aberta, ele fecha-a
    - Imprime se a porta está aberta ou fechada

Exercício 3, Parte 2

- Modifique a classe Prisoner para que…
  - O método display() imprima o mesmo nome da cela
  - O método openDoor() seja removido
- Modifique o método main para que...
  - Cell seja instanciada corretamente
  - O prisioneiro não tente mais abrir a porta da cela
  - Isso testa a capacidade da classe Cell de abrir e fechar sua porta
    - Tente fornecer tanto o código segurança correto quanto o incorreto

Continuando a Desenvolver Este Software

- Atualmente, o método main testa a capacidade da porta de uma Cela ser aberta e fechada de acordo com um código de segurança
- O teste permite-nos confirmar se esse recurso foi devidamente implementado
  - Se o recurso não funcionar, ele deverá ser corrigido
  - Se o recurso funcionar, é seguro incluí-lo como parte de outro recurso
- Uma possível etapa seguinte seria desenvolver uma classe Guard com um método para inserir um código de segurança
  - Definitvamente, uma classe Guard, e não o método main, seria a responsável por inserir o código de segurança

Lembre-se do Modelo Espiral de desenvolvimento.

A Função do Método Main

- Alguns programas são orientados por objetos físicos
- Alguns programas são orientados por botões
- Neste exercício, o método main modela ações que orientariam o programa
  - Chamar `bubba.openDoor()` modela um prisioneiro que está tentando abrir a porta da sua cela
  - Chamar `cellA1.setIsOpen(1234)` modela uma pessoa que inseriu um código de segurança

Exercício 4

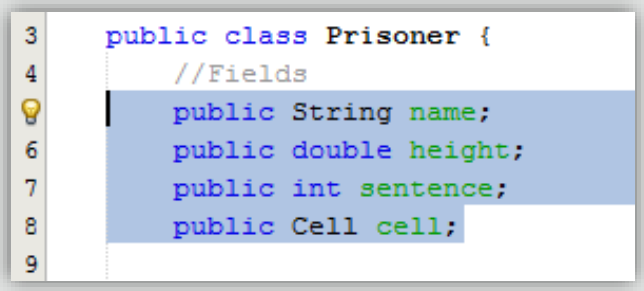
- Continue a editar o projeto `PrisonTest`
- Encapsule a classe `Prisoner`
  - Torne seus campos private
  - Forneça métodos getter e setter para cada campo

Este Exercício Não Foi Divertido!

- O Exercício 4 foi cansativo e fez com que você ficasse entediado?
- Alguns programadores preferem controlar o encapsulamento dos campos por conta própria
- Outros programadores preferem que o IDE faça o trabalho para eles
  - Seu IDE pode encapsular campos para você
  - Os slides a seguir mostram como fazer isso no NetBeans
  - Se você estiver usando outro IDE, consulte a documentação para saber como fazer isso

Truque de Encapsulamento do NetBeans

1. **Realce** os campos que você deseja encapsular

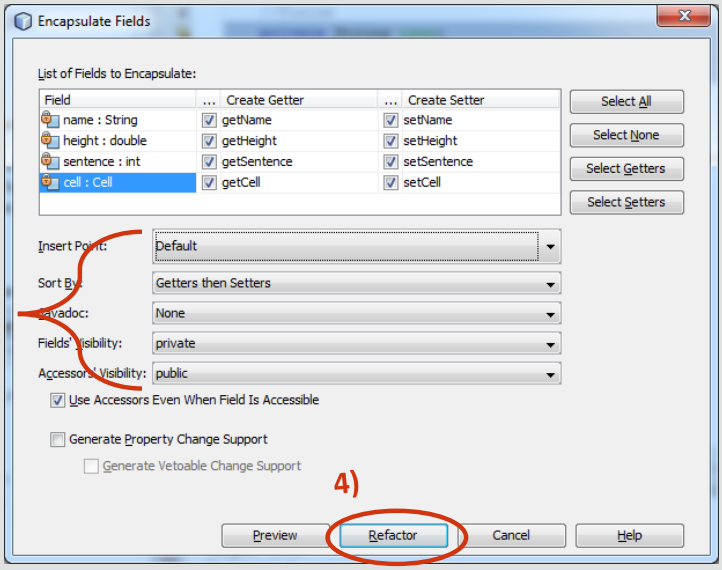


2. Clique com o botão direito do mouse e selecione **Refactor >> Encapsulate Fields**

Truque de Encapsulamento do NetBeans

3. Ajuste as configurações de acordo com suas preferências
4. Clique em Refactor

3) Recomendamos estas configurações



Resumo do Encapsulamento

- O encapsulamento oferece técnicas para limitar a visibilidade de uma classe
- O acesso e a visibilidade devem ser limitados o máximo possível
- A maioria dos campos devem ser private
- Forneça métodos getter para retornar o valor dos campos
- Forneça métodos setter para modificar campos com segurança

Resumo

- Nesta lição, você deverá ter aprendido a:
  - Entender a interação de objetos mais detalhadamente
  - Usar o modificador private para definir variáveis de classe
  - Entender a finalidade de métodos getter
  - Entender a finalidade de métodos setter


ORACLE
Academy