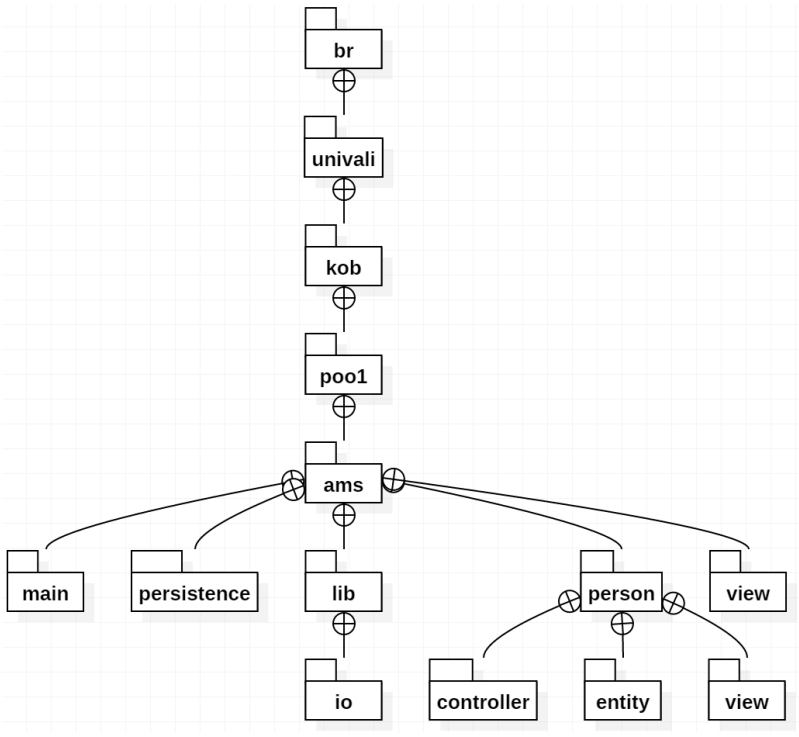
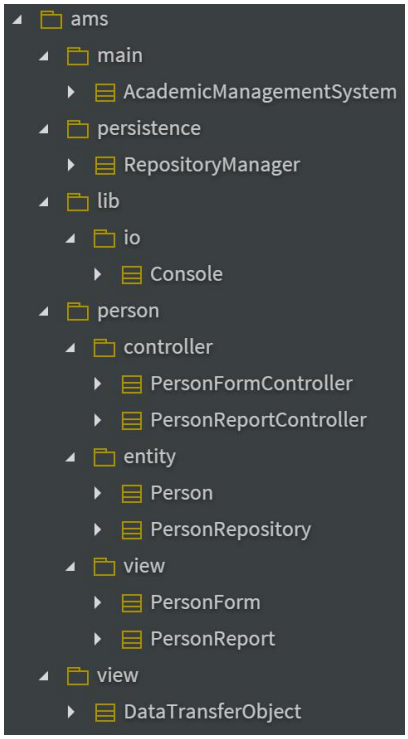
Roteiro 02 – Exercícios de Fixação

1. O que significa a classe **Person** ser uma superclasse de **Student**?
2. O que significa a classe **Student** ser uma subclasse de **Person**?
3. Qual a relação entre o relacionamento de generalização/especialização (herança), abstração e reusabilidade?
4. Faz sentido dizer que um objeto instanciado a partir da classe **Student** tem um objeto **Person** dentro dele (justifique sua resposta)?
5. Qual classe é mais abstrata (**Person** ou **Student**) e por que?
6. Qual classe é mais reutilizável (**Person** ou **Student**) e por que?
7. Qual classe é mais genérica (**Person** ou **Student**) e por que?
8. Para acessar os atributos definidos em **Person**, podemos trocar sua visibilidade para ***public*** ou ***protected***?
9. Qual a diferença entre os tipos de visibilidade (***private***, ***protected***, ***public***, ***package***)?
10. Todas as operações devem ser públicas?
11. Quando utilizamos operações públicas e quando utilizamos privadas?
12. Quando utilizamos operações protegidas?
13. Como garantir que um atributo definido como privado na superclasse possa ser acessado somente pelos métodos de suas subclasses?
14. Qual a diferença entre operação e método?
15. Quais as orientações para dar nome a uma classe?
16. Quais as orientações para dar nome a um atributo?
17. Em quais situações percebemos que a classe não representa uma boa abstração?
18. O que é a assinatura de uma operação?
19. O que é um objeto imutável?
20. Durante a modelagem de um sistema, como identificamos as classes que serão necessárias?
21. Defina coesão e relacione este conceito com OO.
22. Com relação ao último passo da atividade 03 (`PersonListTest`):
    a. Qual conceito fundamental OO foi utilizado para permitir um vetor de `Person` receber objetos `Student` e `Employee`?
    b. Você notou que não foram instanciados objetos `Person`? Por que?
    c. Observando as operações `createStudent(...)` e `createEmployee(...)`, o que poderíamos melhorar na modelagem e implementação das nossas classes de entidade (`Person`, `Student` e `Employee`)?
23. Qual a diferença entre uma classe abstrata e uma classe concreta?
24. Por que (e/ou quando) utilizamos classes abstratas?
25. O que é um padrão de projeto (*design pattern*)?
26. Qual a vantagem de utilizar padrões de projeto (*design patterns*)?
27. Quais as vantagens e desvantagens do padrão *Telescoping*?
28. Por que não implementamos o padrão *Telescoping* na classe `Person`?

Roteiro 02 – Exercícios de Fixação

29. O que é *code smell* (ou *bad smell*)?
30. O que é polimorfismo?
31. Qual a vantagem de utilizar polimorfismo?
32. Qual a relação entre classes abstratas e polimorfismo (este tipo de polimorfismo é chamado de inclusão ou de subtipo)?
33. O que é sobrecarga de operações (classificada como polimorfismo *ad-hoc* ou aparente)?
34. Quais as orientações para o uso adequado de sobrecarga?
35. Apresente suas sugestões para os questionamentos feitos no último passo da atividade anterior. No próximo roteiro, voltaremos a este tópico e você poderá confrontar suas anotações.
36. Descompacte o arquivo "POO1 - MVC Example 01 (Person Form and Report)". Este arquivo contém a a modelagem UML (formato StarUML) e o respectivo código fonte para cadastrar e listar pessoas. A proposta é que ele seja uma referência inicial sobre como separar camadas de abstração (interação com o usuário, controle de fluxo do negócio, processamento, entidades). Na modelagem, existem três diagramas construídos. Um deles apresenta a estrutura de pacotes definida para a aplicação (foco na coesão). Entre em cada pacote e verifique com as classes estão organizadas:



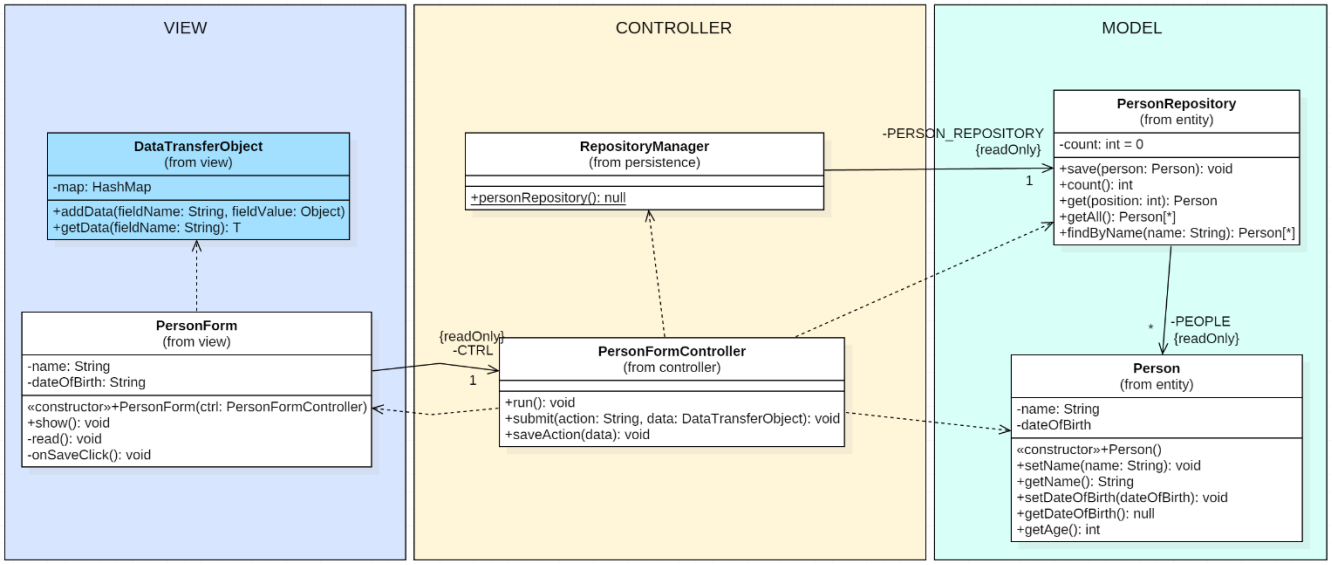


Os outros dois diagramas de classe apresentam as classes envolvidas no cadastramento de pessoas (MVC Person Form) e na listagem de pessoas (MVC Person Report). Observe que as classes foram organizadas no diagrama de modo a facilitar a visualização daquelas relacionadas com cada camada de abstração:

- View – onde ficam as classes de entrada (leitura de dados) e saída (apresentação de dados); onde a interação com o usuário funciona. Conversa com a camada de controle recebendo e passando dados para o devido processamento (feito pelo controle).
- Controller – controla uma funcionalidade, fazendo o meio de campo com a camada onde estão as entidades (model); responsável pelo processamento.

Roteiro 02 – Exercícios de Fixação

- Model – onde as entidades de negócio estão localizadas; foi definida uma classe `Repository`, a qual é responsável pela persistência (armazenamento) efetiva da entidade; ela é similar uma lista e abstrai da classe usuária, a necessidade de conhecer como os objetos são armazenados. Neste exemplo, não estamos implementando persistência de verdade, uma vez que os objetos são guardados em um vetor em memória primária. Porém, futuramente, iremos aprender como fazer isso em memória secundária (disco).



Crie um projeto no NetBeans a partir do código fonte fornecido (pasta `src`). Para visualizar melhor a estrutura de pacotes no NetBeans, na área Projetos, clique com o botão direito em qualquer área sem elementos. No menu de contexto, selecione a opção [Exibir Pacotes Java como | Árvore Reduzida]. Se você optar por [... | Árvore], a estrutura mostrada será exatamente como os pacotes estão estruturados (mesmo que eles não tenham classes ainda).

No caso do cadastro de pessoas (diagrama acima), observe que a comunicação entre as camadas VIEW e CONTROLLER é feita com a utilização de um objeto de transferência de dados (DTO) para transferir os dados preenchidos pelo usuário do formulário para o objeto que controla o cadastro (`PersonFormController`). A interação com o usuário é totalmente via console (veja a classe `br.univali.kob.poo1.ams.lib.io.Console`). Entretanto, o código está organizado para que mudanças futuras na forma de interação (ex: interface gráfica), tenham pouco impacto nas demais camadas da aplicação.

A partir do conteúdo abordado por este roteiro, percebemos que objetos `Person` serão raramente utilizados em uma aplicação. Logo, crie um projeto para cadastrar e mostrar empregados. Adicione funcionalidades, complete o formulário de entrada, faça testes, etc. O objetivo desta exploração é fazer com que você se sinta mais confortável com a linguagem Java e com os conceitos discutidos.

A sequência de passos abaixo mostra como funciona o cadastro de pessoas:

a) Algum objeto (no exemplo apresentado, um objeto `AcademicManagementSystem`) pede para o objeto `PersonFormController` executar sua função principal `run()`. Isso equivale a pedir para iniciar o cadastro de pessoas.
b) O objeto `PersonFormController` cria um objeto de formulário `PersonForm` (este formulário corresponde à tela que interagirá com o usuário).
c) O objeto `PersonFormController` pede para o objeto `PersonForm` se mostrar, invocando o método `show(this)`. O `this` corresponde ao próprio objeto `PersonFormController`. Desta forma, o objeto `PersonForm` poderá retornar os dados fornecidos mais tarde.
d) O objeto `PersonForm` aguarda a digitação dos dados pelo usuário, por meio do método `read()`.
e) Quando o usuário decide gravar os dados, o objeto `PersonForm` invoca o método `onSaveClick()`. Este método cria e povoa um objeto `DataTransferObject` com o valor dos campos preenchidos pelo usuário.

Roteiro 02 – Exercícios de Fixação

f) O objeto `PersonForm` pede para que o objeto `PersonFormController` execute a ação “save” com os dados que foram preenchidos, invocando o método (do controlador) `submit("save", dto)`.
g) O objeto `PersonFormController` verifica qual a ação a ser executada e invoca o método correspondente.
h) Para salvar a pessoa, o objeto `PersonFormController` cria um objeto e preenche os atributos com as informações recebidas do objeto `DataTransferObject`.
i) Por último, o objeto `PersonFormController` pega o repositório de pessoas (`PersonRepository`) com o gerenciador de repositórios (`RepositoryManager`) e solicita que o objeto `Person` criado seja salvo. O gerenciador de repositórios é uma classe que garante o acesso ao repositório correto.