P.PORTO ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

IPP – Instituto Politécnico do Porto

LEI/LSIRC

PP – Paradigmas de Programação

2º Semestre

Ficha Prática 9

Sumário
- Classes Abstractas

Documentação complementar Java
- Classe Object
- Classes abstratas

Nota: A resolução desta ficha prática pressupõe a utilização da ferramenta git para ajudar na gestão de versões do software desenvolvido.
Documentação complementar Git:
- learngitbranching
- gitexplorer

1. Ao longo dos anos, a empresa 4Patas tem disponibilizado um conjunto de serviços muito elogiados pelos seus clientes que privilegiam o tratamento competente e muito cuidado que os diversos colaboradores da empresa proporcionam aos seus animais de estimação. Como resultado de tão reputado serviço, a empresa tem crescido no sentido de disponibilizar mais e melhores serviços aos seus clientes.

Devido ao enorme volume de solicitações, a empresa resolveu partir para a implementação de um pequeno sistema que suporte a gestão de reservas dos seus clientes. Todo o processo inicia-se com o contacto entre o cliente e o colaborador da empresa encarregue de registar as reservas. O cliente informa o colaborador, indicando os dados do animal (identificador, nome, género e idade estimada) e os serviços que pretende que o animal receba.

Atualmente a empresa disponibiliza os seus serviços para cães e gatos, não excluindo a possibilidade, de num futuro próximo, abranger outro tipo de animais. Para os cães, a empresa tem a necessidade de armazenar o seu porte (pequeno, médio ou grande) e para os gatos é necessário identificar o tipo de gato que irá receber: gato de interior ou gato de exterior.

O colaborador começa por registar a data da reserva e os dados do animal, questionando posteriormente o cliente relativamente aos serviços que pretende incluir da lista que a empresa disponibiliza: Passeios, Alojamento, Transporte, Banho e tosquia e sessão fotográfica. Após a conclusão do serviço, o registo da reserva transita para o histórico de reservas concluídas.

A gestão de reservas deverá disponibilizar os seguintes serviços:

- Adicionar reservas, considerando que não podem existir duas reservas iguais armazenadas (para o mesmo animal numa determinada data). O método deverá retornar um valor que sinaliza o sucesso/insucesso da operação.
- Validar uma reserva recebida, verificando se esta existe e identificando-a como reserva concluída. O método deverá retornar um valor que sinaliza o sucesso/insucesso da operação;
- Disponibilizar uma representação textual das reservas que ainda não foram satisfeitas numa determinada data.

Para suportar a gestão de reservas, deverá tirar partido da classe `ContainerOfObjects` implementada na ficha prática anterior. Deve utilizar a sua classe incluindo-a como uma biblioteca no novo projeto.

Para isso:
- Crie um novo projeto com o nome: `ContainerOfObjectsAPI` e copie apenas a classe `ContainerOfObjects`. De seguida, clique com o botão direito rato e selecione: `clean and build` sobre o projeto.
- Na pasta do projeto foi criado um ficheiro com a extensão `.jar` que se encontra na pasta `dist`;
- O ficheiro com a extensão `.jar` representa as classes contidas no projeto e pode ser utilizado

como uma biblioteca que disponibiliza a estrutura até agora criada, para novos projetos.

- Importe o ficheiro com a extensão `.jar` para as bibliotecas do seu novo projeto.
- Dentro da pasta do seu novo projeto crie uma pasta chamada `libraries` e coloque o ficheiro com a extensão `.jar` nessa pasta;
- Clique com o botão direito do rato no seu novo projeto e selecione: `Properties`;
- Na nova janela, do lado direito em `categories`, selecione `libraries` e no novo painel selecione o botão `Add JAR/folder`;
- Selecione o ficheiro que colocou na pasta `libraries` do seu projeto.

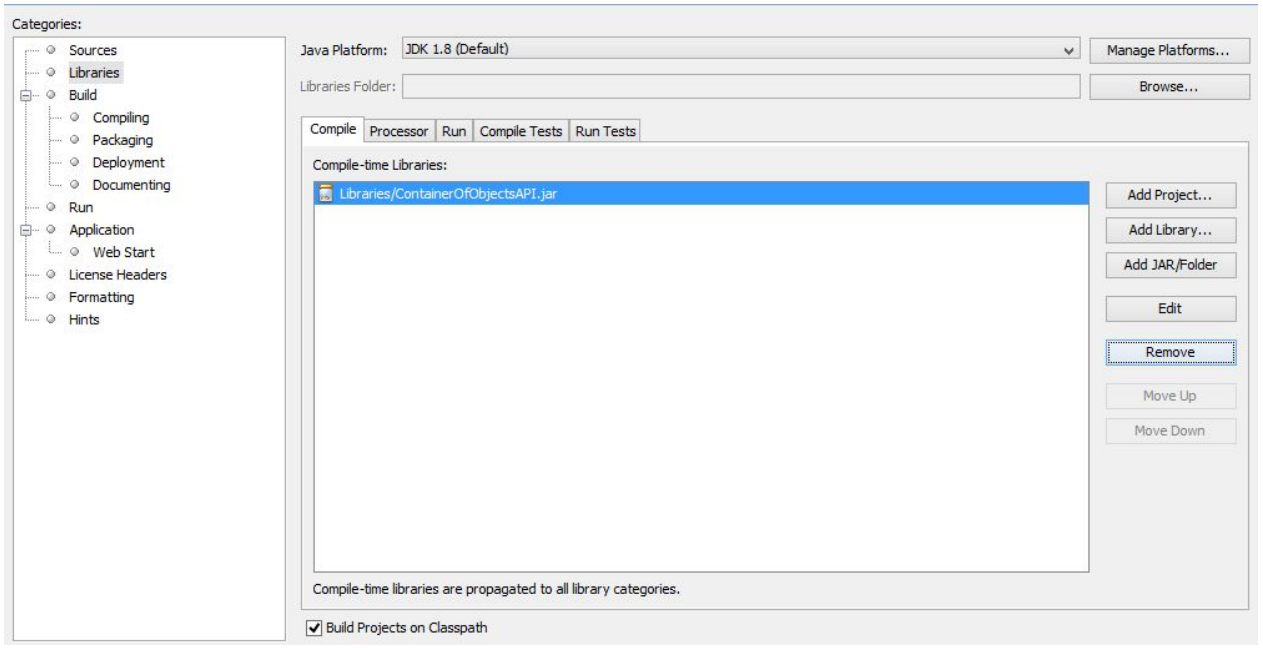

Figura 1. Exemplo da importação da biblioteca `ContainerOfObjects`

Utilize um repositório git para suportar o desenvolvimento do exercício, realizando *commits* regulares que acompanhem o desenvolvimento das várias componentes.

Teste o código através de uma classe `Demo` que deverá ser criada num `package` específico. Recorra a comentários JavaDoc, e não só, de modo a documentar, o mais exaustivamente possível, o código que desenvolveu.

2. Uma pequena instituição do ensino superior pretende criar uma aplicação que permita gerir todas as pessoas envolvidas no normal funcionamento de uma escola, nomeadamente os alunos e colaboradores (Professores e pessoal administrativo/suporte). Cada tipo pode ser caracterizado por:

- **Professor**, caracterizado por: Código de identificação (por exemplo: tsm), Nome, Nif, Morada, Telefone, habilitações literárias (Licenciatura, Mestrado ou Doutoramento), área científica para a qual foi contratado (Informática, Biomedicina, etc), data de contratação e a data de fim de contrato.
- **Aluno**, caracterizado por: Código de identificação (por exemplo: 808080), Nome, Nif, Morada, Telefone, data da primeira matrícula e tipo (que identifica se é ou não trabalhador estudante).
- **Pessoal administrativo/suporte**, caracterizado por: Código de identificação (por exemplo: tsm), Nome, Nif, Morada, Telefone, habilitações literárias (Licenciatura, Mestrado ou Doutoramento), área a qual se encontra associado (por exemplo: recursos humanos, secretaria, etc), data de contratação, percentagem do contrato e a data de fim de contrato.

Todas as pessoas na instituição possuem **obrigatoriamente** um número de horas semanais em que se encontram na instituição. O cálculo das horas é realizado em função do cargo que a pessoa ocupa:

- **Professor**: Cada professor pode lecionar mais do que uma **disciplina** (caracterizado por Código e nome) e uma disciplina pode ser lecionada por mais do que um professor. O número total de horas é calculado pela soma do número de horas a que o professor se encontra associado em cada disciplina. Um professor não pode lecionar mais de 5 disciplinas.
- **Aluno**: Cada aluno pode frequentar um conjunto de disciplinas de um **curso** (caracterizado por

Código e nome). O número de horas é a soma total das horas semanais (carga horária) de cada disciplina a que se encontra inscrito. Um aluno não pode estar inscrito a mais de 10 disciplinas.
- **Pessoal administrativo/suporte**:O número de horas é calculado em função da percentagem do contrato (100% equivale a 40 horas semanais).

Utilize um repositório git para suportar o desenvolvimento do exercício, realizando *commits* regulares que acompanhem o desenvolvimento das várias componentes.

1.1 Crie a estrutura de classes necessária para representar o problema e teste utilizando uma classe apropriada.

1.2 Crie uma classe: `PersonManagement` para permitir a gestão de pessoas da instituição. A classe `PersonManagement` deverá ser capaz de adicionar, remover e listar pessoas. Teste a classe implementada.

1.3 Implemente uma classe `CourseManagement` para gerir as disciplinas associadas aos alunos e professores. Note que no caso dos professores, deverá ser armazenado o número de horas que o professor leciona para uma determinada disciplina. Deverá ser possível:

- Adicionar disciplinas a um professor e aluno (não devem existir cópias do mesmo objeto). Não deve ainda permitir que existam disciplinas repetidas para o mesmo professor/aluno;
- Remover disciplinas a um professor e aluno;
- Imprimir disciplinas de um professor e aluno.

1.4 Recorra a comentários JavaDoc, e não só, de modo a documentar, o mais exaustivamente possível, o código que desenvolveu.