Aula Prática 3

Resumo:
- Programação modular.

Exercício 3.1

Recupere a classe `Data` desenvolvida no exercício 2.4 e acrescente-lhe:

- Um construtor que inicie a data a partir de uma string no formato ISO (como `"2018-02-28"`). Se a string for inválida, o construtor deve indicá-lo e terminar o programa (com `exit`).[1]
- Um método `compareTo` para comparar datas. O resultado de `x.compareTo(y)` deve ser um inteiro nulo, positivo ou negativo consoante `x` seja igual, maior ou menor que `y`, respetivamente.

Utilize o programa `TestData2` para testar esta funcionalidade.

Exercício 3.2

Experimente executar o comando: `java -jar SortDates1.jar dates1.txt`. Esse programa lista, por ordem cronológica, um conjunto de datas lidas do ficheiro dado no argumento.

`SortDates1.java` é uma implementação incompleta e com vários erros desse programa. Comece por corrigir os erros sintáticos até conseguir executá-lo. Depois analise os erros de execução, identifique e corrija as suas causas. Finalmente, descomente a chamada à função de ordenação e teste de novo para detetar e corrigir os erros restantes.

Exercício 3.3

Crie uma classe `Tarefa` que permita representar um texto associado a um intervalo entre duas datas. Utilize o programa `TestTarefa1.java` para inferir qual tem de ser a *interface* da classe, isto é, o conjunto dos seu membros públicos. Teste usando o comando `java TestTarefa1 tasks1.txt`.

[1] Veremos formas melhores de lidar com falhas noutra aula.

Exercício 3.4

Utilizando as classes desenvolvidas nos exercícios anteriores construa uma nova classe `Agenda` onde seja possível registar até mil tarefas. Nos objetos deste tipo deve ser possível realizar as seguintes operações:

Método `novaTarefa`: Acrescenta uma nova tarefa à agenda. As tarefas devem ser mantidas por ordem crescente das suas datas iniciais.

Método `escreve`: Mostra o conteúdo completo da agenda.

Método `filtra`: Extrai para uma nova agenda as tarefas que intersectam um certo intervalo de datas. Sugestão: acrescente à classe `Tarefa` um método `t1.intersecta(t2)` que permita verificar se duas tarefas se intersectam.

Teste a classe com o programa `TestAgenda.java`. Compare o resultado com o do comando `java -jar TestAgenda.jar`.

Exercício 3.5

Uma empresa de construção precisa de gerir a informação, bem como extrair diversas propriedades, das habitações que constrói.

Desenvolva um conjunto de classes para esta aplicação. Fornece-se em anexo o programa `TestHouses`, que permite testar as funcionalidades pretendidas, bem como a classe `Point` usada para representar pontos num espaço cartesiano.

a. Desenvolva uma classe `Room` para representar as divisões das habitações. Considera-se que cada divisão tem uma forma rectangular, alinhada com os eixos de um determinado sistema de coordenadas. Esta classe deverá ter os seguintes métodos públicos:

   - Construtor com três argumentos, nomeadamente o tipo da divisão (uma cadeia de caracteres), e as coordenadas dos cantos inferior esquerdo e superior direito;
   - `roomType()` - devolve o tipo da divisão;
   - `bottomLeft()` - devolve o canto inferior esquerdo;
   - `topRight()` - devolve o canto superior direito;
   - `geomCenter()` - devolve o centro geométrico da divisão;
   - `area()` - devolve a área da divisão.

b. Desenvolva uma classe `House` para representar as habitações, com os métodos:

   - `House(String)` - Construtor que recebe como argumento e regista o tipo da habitação (uma cadeia de caracteres que poderá ser `"house"` ou `"apartment"`); além disso, este construtor deve reservar memória para 8 divisões e deve também registar que, caso venha a ser preciso armazenar informação sobre mais divisões, a memória das divisões será expandida em blocos de 4 divisões adicionais (inicialmente 8, depois sucessivamente 12, 16, etc., conforme as necessidades);

- `House(String, int, int)` - Construtor que recebe como argumentos o tipo da habitação, o número de divisões para as quais se vai inicialmente reservar memória, bem como o número de divisões adicionais a reservar sempre que a memória esteja cheia;
- `addRoom(Room)` - adiciona uma nova divisão à habitação;
- `size()` - devolve o número de divisões da casa;
- `maxSize()` - devolve o número máximo de divisões que é possível armazenar num dado momento;
- `room(int)` - dado um índice de uma divisão (um inteiro entre `0` e `size()-1`), devolve a divisão correspondente;
- `area()` - devolve a área total da habitação, dada pela soma das áreas das divisões;
- `getRoomTypeCounts()` - devolve os tipos de divisões existentes com o número de divisões de cada tipo, na forma de um vector (array) de elementos da seguinte classe:

```
public class RoomTypeCount {
    String roomType;
    int count;
}
```

  Nota: este vector não deverá estar sobre-dimensionado.
- `averageRoomDistance()` - devolve a distância média entre as divisões da casa, tomando como referência os respectivos centros geométricos;

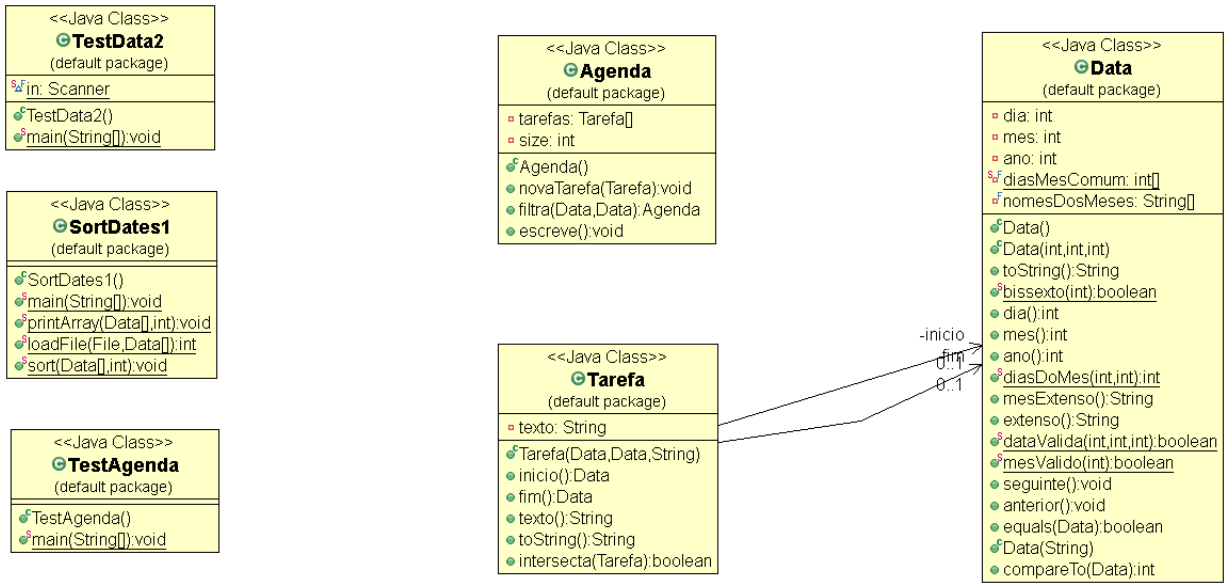

Figura 3.1: Diagrama das classes dos exercícios 3.1, 3.2, 3.3 e 3.4.