Projeto de Programação Orientada para Objetos 2014 - 2015

Simulador de consumo eléctrico

1. Introdução

A recente criação de aparelhos domésticos que comunicam juntando-se à *Internet of Things* (IoT) e de contadores de energia "inteligentes" permite construir aplicações a pensar nestes sistemas. O trabalho proposto é uma simulação de um ambiente doméstico em que há vários aparelhos a contribuir para o consumo de energia. Queremos por um lado parametrizar a simulação (quais os aparelhos ligados, onde e por que períodos) e depois queremos ver o gráfico resultante do consumo por ramal.

2. Funcionamento geral (versão final)

Deve implementar um sistema de simulação de consumo elétrico, com as seguintes funcionalidades.

1. Ao iniciar o programa deve **processar toda a informação gravada nos ficheiros**

   ```
   instalacao.json,
   ligacoes.json,
   aparelhos.json,
   eventos.json.
   ```

   A leitura é feita pelo código dado no main() do código fornecido. O processamento dos dados resultantes da leitura deve ser feito de modo genérico para quaisquer ficheiros com o formato indicado. Ao processar devem ser criados os objectos das classes que representam cada um dos conceitos importantes na resolução do problema e as ligações entre estes;
2. Após ler os ficheiros, **o simulador deve correr iterativamente desde o momento zero até atingir o ciclo correspondente ao tempo de simulação pedido pelo utilizador.** Cada iteração do ciclo simulará uma unidade de tempo a passar; Em cada iteração devem ser simulados os eventos que devem ocorrer nesse momento. Os eventos devem ser guardados numa lista (ou fila) e retirados por ordem crescente de tempo.
3. **Durante a simulação deve ser apresentado, usando a classe Chart que implementa um Observer,** um gráfico do consumo por linha, tal como no exemplo de execução em anexo (as linhas e valores no exemplo foram geradas com ficheiros de configuração de teste que podem ser diferentes dos distribuídos. A última versão do projeto-base estará sempre no e-learning em Conteúdos da Unidade Curricular / Enunciado do Trabalho Final / TFPOO2015Vx_x.zip, importe no eclipse usando Import / Existing Projects Into Workspace / Archive File ...). Para actualizar o gráfico deve chamar os métodos da classe Observable do seguinte modo:

   ```
   Map<String, Double> potencias = ... ; // Mapa que associa o nome de cada
   linha com a sua potência actual.
   // ... TODO
   setChanged();
   notifyObservers(potencias);
   ```

O gráfico deve ser registado como observador da instalação no Main:

Essa é a única forma permitida de actualização do gráfico.

Pf não altere o formato das linhas ou outros parâmetros do gráfico. A classe `Chart` tem documentação anexa ao projecto, em `doc/pt/iscte/poo/powerhouse/chart/Chart.html`. O padrão de desenho Observer - Observable será assunto de explicação detalhada em aula.

3. Formato dos ficheiros de configuração

A leitura e processamento dos ficheiros JSON deve ser feita usando as classes do pacote json-simple, já incluído no projecto. Podem encontrar exemplos de utilização aqui.

Aconselha-se a que a descodificação seja progressiva passando os objectos `JSONObject` como argumentos de construção de cada um dos construtores a invocar. Note que:

- a ordem dos parâmetros nos ficheiros não é, em geral, relevante
- os espaçamentos e *tabs* são ignorados pelos métodos da biblioteca JSON-simple
- quase todos os parâmetros são opcionais nos aparelhos (excepto tipo e id) sendo usados apenas onde necessário. Nos eventos os parâmetros "valor" e "programa" também são opcionais.

Segue-se o formato dos ficheiros, há um exemplo de cada um deles no projecto, a partir da V2.

Instalação (instalacao.json):

```
[
   {
    "nome": <nome da linha>,
    "tomadas": <número de tomadas>
   },
  ...
]
```

Aparelhos (aparelhos.json):

```
[
      {"tipo" : <tipo de aparelho*>,
       "id": <identificador único do aparelho>,
        "potenciaMaxima": <valor, double, da potência máxima caso seja um aparelho de potência variável>,
       "potencia" : <valor da potência fixa, caso seja um aparelho de potência fixa>,
       "nTomadas" : <número de tomadas caso seja uma tripla>,
       "programas" : [
            { "id": <nome do programa>,
              "ciclos" : [
                  { "t" : <tempo desta fase do programa>,
```

```
                    "potencia":  <potência desta fase do programa},
                ...
    }, ...

]
```

O programa de uma máquina de lavar tem um identificador e uma lista de ciclos que se devem seguir na sequência apresentada. Cada ciclo tem o tempo que dura aquela etapa e a potência usada nessa fase.

A tripla não é um aparelho, apenas implementa o interface `Ligavel` e ao ser adicionado a uma linha "acrescenta" a essa linha o número de tomadas indicado no parâmetro `nTomadas`. Ao ser removido retira as suas três tomadas da linha. A inclusão da tripla como subclasse do Aparelho, e não apenas Ligavel, será permitida como simplificação.

O computador terá, em cada iteração, uma potência aleatória entre 0 e a potência máxima.

A lâmpada varíável tem uma potência variável entre 0 e potência máxima.

Eventos (eventos.json):

```
[
  {"accao"  :  <acção,  valores  possíveis:  LIGA,  DESLIGA,  AUMENTO,
DIMINUICAO, PROGRAMA>,
 "idAparelho" : <id aparelho>,
 "tempo" : <momento em que deve ocorrer a acção>},
  "valor"  <valor  da  alteração  caso  a  acção  seja  AUMENTO  ou
DIMINUIÇÃO>,
  "programa" : <id do nome do programa> }, ...
]
```

4. Requisitos de implementação

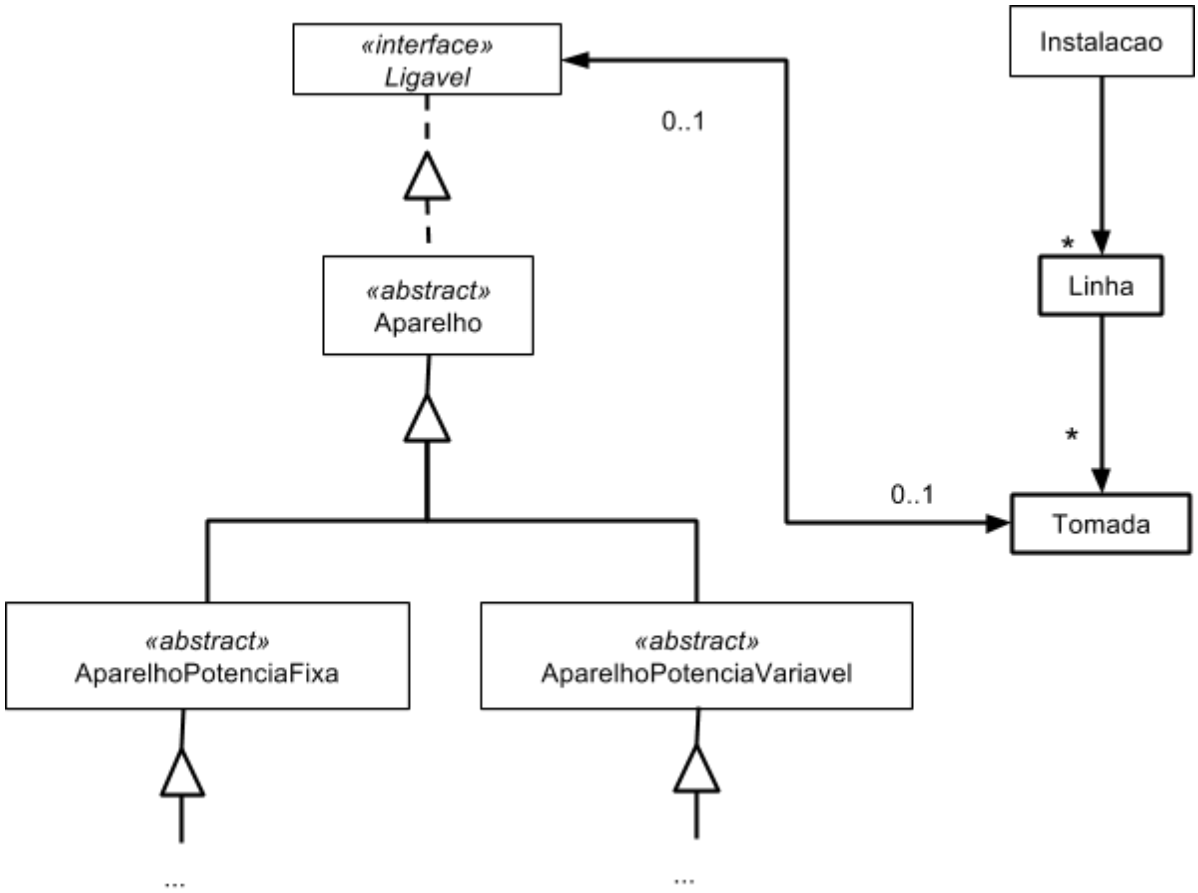

Figura 1. Desenho parcial, de alto nível, das relações entre as principais classes da aplicação. Uma das classes Linha ou Tomada (mas não ambas) pode não existir desde que sejam mantidas as funcionalidades que lhe dizem respeito.

1. Têm de ser **adequadamente usados os interfaces fornecidos** no projeto exemplo.
2. Deve ser usada a classe `Chart` para dar informação visual sobre os consumos por linha (ramal) ao utilizador.
3. As **JUnits de teste fornecidas (até 11 de Abril) deverão estar a funcionar**.
4. As **excepções verificadas (aquelas cujo tratamento é obrigatório) devem ser tratadas ao nível adequado** à sua recuperação sempre que possível, ou de modo a que o utilizador seja avisado da causa do erro antes do programa terminar. **O programa não deve terminar abruptamente sem aviso prévio.**
5. As classes fornecidas devem ser completadas a partir dos métodos indicados com os comentários TODO (leia-se, em inglês, "*to do*"), mas devem ser evitadas outras alterações às classes dadas no projecto-base (excepto para alterar o modo como as excepções são tratadas). Mude logo que possível o nome do seu projecto de modo a incluir no nome o nome e número dos elementos do grupo, por exemplo:
   `TFPOO2015_JosePereira_12345_JoanaCosta_23456`
6. Pode adicionar as classes que entender, respeitando os interfaces e o desenho na figura 1.
7. **As classes devem estar adequadamente distribuidas por *packages,* de modo a serem facilmente transferidas para outro projeto, dado que poderão ser fornecidas várias versões do projeto-base;**

8. Devem criar um **relatório** sumário (em PDF, máximo 4 páginas, incluído no projeto, na pasta base) contendo:
    a. A identificação do grupo;
    b. Máximo uma página sobre se o trabalho atinge os objetivos e quais as opções tomadas (opcional);
    c. Um desenho de alto nível (apenas relações entre as classes principais) em UML (baseado na figura 1);
    d. Uma declaração do grupo de que todo o código apresentado é exclusivamente da sua própria autoria.
9. **Deve existir uma hierarquia como indicado na figura 1**. Esta hierarquia deve ser usada para simplificar o funcionamento do programa e a clareza do código. É fundamental que os métodos sejam declarados e implementados ao nível certo, tipicamente na classe dos dados que usam.

5. Sequência de funcionamento e responsabilidades das classes

1. Ao iniciar a aplicação, devem ser lidos todos os ficheiros de configuração e criadas as estruturas de objectos correspondentes.
   Atenção ao seguinte:
    - O código dado no `Main` já faz a leitura dos ficheiros para objectos do tipo `JSONObject`. O seu trabalho será descodificar os objectos `JSONObject` e gerar os objectos do seu projecto com os argumentos correctos;
    - Deve verificar possíveis problemas de formato e definir o tratamento das exceções que podem ser lançadas;
2. É necessário pedir ao utilizador para inserir o tempo de fim da simulação, isso pode ser feito pela consola usando o `Scanner` ou numa janela simples usando o método, `JOptionPane.showInputDialog();`
3. Deve ser iniciado gráfico que irá ser construído durante a simulação;
4. Deve ser simulado o funcionamento da casa desde o momento `t = 0` até ao tempo final dado pelo utilizador, em cada tempo deve ser registada, para cada linha, a potência total consumida (note que terá de "percorrer" as linhas e tomadas e saber que aparelho está ligado a cada uma das tomadas e qual a potência que está a consumir).

6. Requisitos de entrega

O não cumprimento dos requisitos numa das entregas pode implicar a eliminação do aluno / grupo do processo de avaliação, só podendo voltar a tentar obter aprovação à disciplina em Época Especial, se admissível (entregando novo projeto) ou no próximo ano letivo.

A decisão sobre o cumprimento ou não dos requisitos é do docente que acompanhou o trabalho, tal como no caso de uma entrega, intercalar ou final, de um projeto real, um cliente pode decidir que as condições do contrato não foram cumpridas e terminar o projeto abandonando a encomenda feita.

Versão final:

1. **Deve entregar a versão final do trabalho até às 08:00 de 25 de Maio de 2015**;
2. Os alunos cujo trabalho foi acompanhado por um docente devem entregar na pasta do docente no e-learning, em "**Conteúdos da Unidade Curricular"** na pasta "**Entrega do Trabalho Final 2014 - <Nome do Docente>"**, **e enviar por mail para o docente que acompanhou o trabalho**;
3. Se nenhum docente acompanhou o trabalho por não frequentar habitualmente as aulas, deve entregar, em "**Conteúdos da Unidade Curricular"** na pasta "**Entrega do Trabalho Final 2015 - Sem Docente" e enviar cópia por mail para o coordenador da UC (luis.nunes@iscte.pt)**;
4. Não deve usar símbolos acentuados no código do trabalho, nem mesmo nos comentários. Obviamente isto não se aplica ao relatório;
5. **O *projecto-eclipse* deve conter todos os ficheiros a entregar: o código fonte das classes nos respectivos packages, as JUnit, os ficheiros de configuração fornecidos na última versão do projecto-base, bem como o relatório, jar executável e todos os restantes ficheiros que quiser a entregar.**
6. Deve entregar apenas um pacote que possa ser automaticamente importado para o Eclipse com Import/Existing Projects into workspace/ Archive File (teste esta importação num computador diferente daquele onde fez o trabalho). Deve exportar usando Export / Archive File. Teste o pacote, importando-o para um *workspace* novo;
7. O nome do projecto e do zip deve incluir o nome e numero dos membros do grupo (use Refactor/Rename para mudar o nome do projecto).
8. Caso não consiga enviar por mail o zip, altere a extensão do ficheiro para "xip". Não use um compactador que gera ficheiros *.rar*.

7. Avaliação

1. O trabalho pode ser feito individualmente ou aos pares (grupos de dois elementos). Não são autorizados grupos de mais do que 2 elementos;
2. Os trabalhos serão verificados em termos de plágio e, além de obviamente serem anulados os trabalhos plagiados (o que não permitirá fazer a UC neste ano letivo), serão feitas queixas ao Conselho Pedagógico sempre que os resultados da análise indiquem claramente que houve plágio de modo a que a penalização seja efetiva;
3. Os trabalhos serão avaliados pelo cumprimento dos requisitos funcionais e de implementação. Deverá apontar para que todos os requisitos sejam cumpridos, tal como num contrato normal com um cliente.
4. Todos os trabalhos deverão ter uma discussão obrigatória, embora grupos e/ou alunos específicos possam ser dispensados caso o acompanhamento do trabalho nas aulas dê ao docente segurança suficiente para comprovar que o aluno é responsável pelo trabalho;
5. Trabalhos que não cumpram vários dos requisitos não permitirão ao aluno o acesso à prova da disciplina e por isso implicam a reprovação à disciplina.
6. *Checklist* de requisitos da entrega final:
    - Pedir ao utilizador para indicar o tempo de fim da simulação
    - Respeitar o desenho de classes da Figura 1
    - Desenhar o gráfico de distribuição de potência adequado para cada conjunto de

ficheiros de configuração que seja dado, dentro dos formatos definidos no enunciado.
- Boa utilização dos interfaces
- Utilização de uma fila prioritária ou lista ordenada para guardar os eventos e implementação do interface *Comparable* na classe que implementa o Evento (ou de um *Comparator* apropriado) de modo a que a fila devolva sempre o evento com o menor tempo na fila.
- Boa distribuição do código, mantendo sempre que possível o código na classe que contém os atributos usados
- Tratamento adequado das excepções (pelo menos as excepções verificadas)
- Documentação em Javadoc de uma classe, pequena
- Passagem nos testes em todas as JUnits fornecidas ao longo do ano
- Entrega de relatório em PDF, na raíz do projecto
- Projecto correctamente exportado para .zip

7. **As normas quanto ao cumprimentos dos requisitos de entrega serão rígidas, o programa deve cumprir TODOS os requisitos acima para garantir a aprovação.**