

**UNIVERSIDADE FEDERAL DE MINAS GERAIS -- ICEx, DCC**  
**PROGRAMAÇÃO MODULAR**

Documentação do Trabalho Prático 1 - Estacionamento Modular

Alunas:       Lívia Almeida Barbosa  
              Tarsila Bessa Nogueira Assunção

## 1. Introdução

O primeiro trabalho prático da disciplina de Programação Modular tem o intuito de apresentar e familiarizar o aluno com os conceitos fundamentais de Orientação a Objetos, com o auxílio da linguagem de programação Java.

A especificação do trabalho define que deverá ser implementado um sistema responsável pela automatização de estacionamento em um prédio. Foram dadas as definições dos tipos de vaga que existem, da quantidade de vagas por andar, do preço por hora de cada tipo de vaga e do formato de entrada e saída.

Algumas características do estacionamento foram deixadas em aberto, as quais são devidamente definidas na seção a seguir.

## 2. Implementação

### 2.1. Decisões de implementação

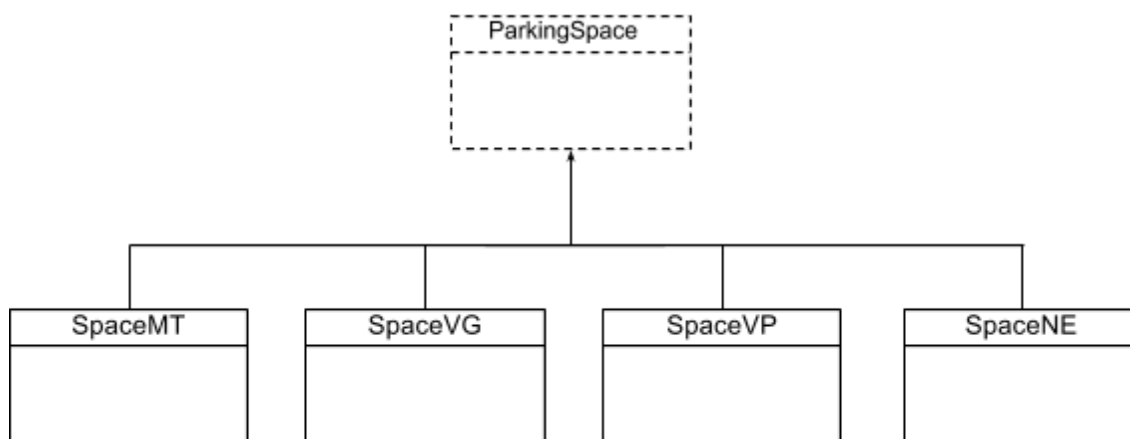
Para utilizar os principais conceitos de modularização e encapsulamento, foi decidido implementar diferentes classes incluídas dentro de um pacote. O pacote criado, `parkinglot` é composto pelas seguintes classes:

- **Constants:** contém toda as constantes utilizadas no programa, como o número de cada tipo de vaga por andar e seu respectivo preço;
- **ParkingSpace:** uma classe abstrata que modela os membros de cada tipo de vaga;
- **SpaceMT, SpaceVG, SpaceVP, SpaceNE:** as classes de cada tipo de vaga, que herdam de `ParkingSpace`, com suas características próprias (tipo, preço) definidas;
- **Floor:** uma classe `Floor`, que é composta por uma lista de objetos da classe `ParkingSpace`, e também possui um atributo com o nível de cada andar e métodos que controlam a entrada e a saída de carros;
- **Algorithm:** classe com o algoritmo que analisa uma linha da entrada e verifica se o veículo pode estacionar no prédio ou não;
- **ParkingLot:** a classe principal, que faz a leitura da entrada e chama o método de verificação de vagas no prédio.

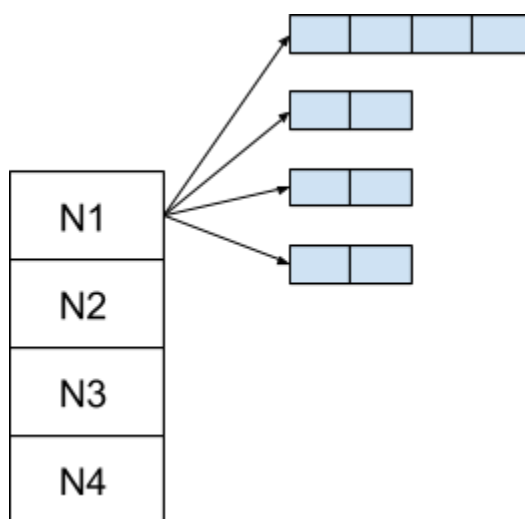
Para guardar as informações relativa às coleções, foram escolhidas as classes `Array` e `ArrayList`, que implementam a interface `Collection<E>`;

Por questões de representação, foi decidido implementar o prédio como Array, pois são espaços contíguos na memória, mais semelhante a um prédio, que possui andares, um acima do outro. As vagas de cada andar foram alocadas como ArrayList, uma estrutura, que é um objeto e de certa forma mais flexível.

Como exemplificação das nossas escolhas para a implementação do trabalho, apresentamos as figuras abaixo. A Figura 1 apresenta uma representação visual da superclasse ParkingSpace e suas subclasses SpaceMT, SpaceVG, SpaceVP, SpaceNE. A Figura 2 apresenta uma representação dos níveis de estacionamento do prédio e suas respectivas vagas.



**Figura 1: Hierarquia de classes**



**Figura 2: Níveis e vagas**

Nos arquivos de entradas de teste, assumiu-se que não haverão veículos com placas clonadas e que, também, os horários de saídas sempre serão após os horários de entrada. Com relação ao preço a ser pago caso o veículo permaneça menos de um hora, será calculada a fração correspondente ao tempo que este permaneceu no estacionamento.

### 2.1.1. Algoritmo para a escolha de vagas

Foi definido na especificação deste trabalho prático que a ocupação das vagas seria dada da seguinte forma:

- As vagas devem obrigatoriamente ser preenchidas sequencialmente do nível mais baixo até o nível mais alto;
- Motocicletas: podem parar em qualquer tipo de vaga (exceto deficientes);
- Veículos pequenos: podem parar em vagas de veículos pequenos ou grandes (exceto deficientes);
- Veículos grandes: podem parar apenas em vagas de veículos grandes;

Desse modo, o algoritmo que processa a entrada de um veículo, verifica se existe vaga para veículos daquele tipo; caso não exista, serão procuradas outras possíveis vagas ainda no mesmo andar. Se também não houver, será procurado no nível superior na mesma ordem, dando preferência para as vagas do mesmo tipo do veículo. Se não existir nenhuma possível vaga no prédio todo, será impresso “LOTADO” para aquela entrada, no arquivo de saída.

A ordem de preferência para os veículos será:

- Veículos pequenos: veículos pequenos, veículos grandes;
- Motos: motos, veículos pequenos, veículos grandes;
- Veículos Grandes: veículos grandes;
- Necessidades Especiais: necessidades especiais, veículos pequenos, veículos grandes;

Foram feitos testes que verificam estes e outros casos, como a saída de veículos que não se encontravam no estacionamento e a entrada após o fechamento do estacionamento. No primeiro caso, será impresso “Carro Inválido” e no segundo, “Horário de Funcionamento: 06:00 às 21:00”.

## 2.2. Descrição de utilização

Na pasta raiz devem ser colocados os arquivos de teste, nomeados como `entrada.txt`. A pasta contém um arquivo Makefile, o qual permite a compilação de todos os módulos automaticamente. Para executar a aplicação, abra um terminal e navegue até a pasta raiz. Em seguida, execute as instruções:

```
$ make
$ make run
```

E a aplicação será executada.

### 2.3. Testes

Para facilitar a execução de testes, foram gerados arquivos de entradas aleatórias para cada teste a partir de uma implementação em Python. Em seguida, analisou-se a saída gerada por cada arquivo criado. Estes estão anexados na seção 5. A seguir, são descritos os casos que foram priorizados em cada tipo de teste.

#### ***Teste 1: Prioridade de estacionamento***

Apresentado na Figura 4, foi considerado o caso no qual todas as vagas de um tipo, já estão preenchidas e foi verificado a ordem no qual as outras vagas do mesmo nível ou dos níveis superiores serão preenchidas. Analisando a saída gerada, verificou-se que a alocação de vagas foi feita de acordo com a prioridade definida anteriormente na seção 2.1.1.

#### ***Teste 2: Veículos inexistentes***

Caso no qual é dada a saída de um carro que não estava presente no estacionamento. A resposta esperada e dada pelo programa foi “Carro Inválido”. A execução desse teste é apresentada na Figura 3.

#### ***Teste 3: Horário de funcionamento***

Mostrado na Figura 3, verifica as situações nas quais os veículos dão entrada antes das 06h e após as 21h. Nessa situação, a resposta obtida do algoritmo, no arquivo de saída, é “Horário de Funcionamento: 06:00 às 21:00”, como esperado.

#### ***Teste 4: Lotação***

Análise do preenchimento de todas as vagas disponíveis, de acordo com a ordem de prioridade de cada tipo de veículo. Neste caso, o algoritmo imprimirá “LOTADO!” no arquivo de saída, como definido na especificação do trabalho. O comportamento descrito pode ser verificado na Figura 5.

## 3. Conclusão

Durante a implementação do trabalho prático, verificou-se a aplicação dos diversos conceitos de Orientação a Objetos, como os conceitos de objetos, herança, polimorfismo, modularidade obtida através das classes e abstração, por exemplo.

Devido ao conhecimento básico da linguagem Java e de Orientação a Objetos, certa parte pertencente ao algoritmo de alocação de vagas não está suficientemente genérico, apresentando grande reuso de código no contexto do algoritmo. Além disso, foi avaliado posteriormente que, em certos métodos, o tipo de retorno deveria ser modificado de int para bool.

Com o trabalho, também foi possível familiarizar com a IDE Netbeans, a qual foi bastante utilizada durante o desenvolvimento do código.

#### 4. Bibliografia

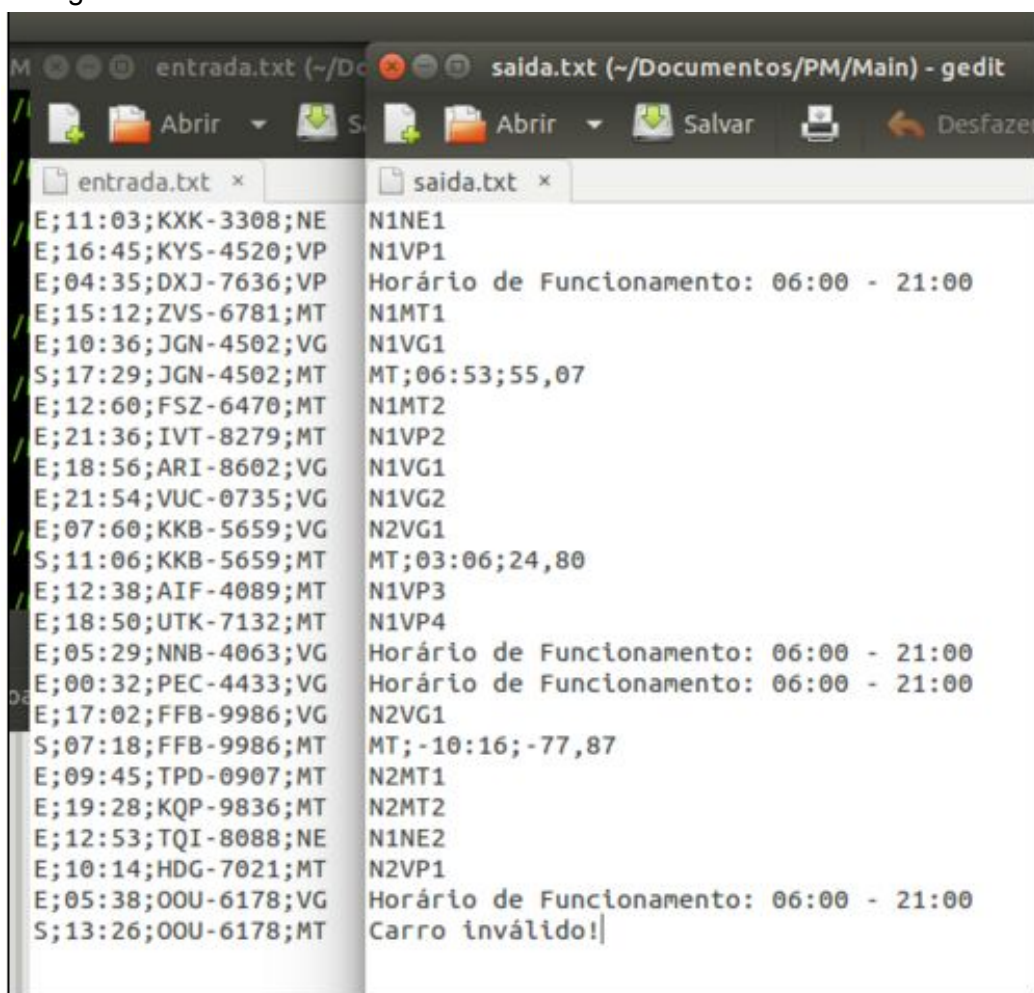
DevMedia. Disponível em: <http://www.devmedia.com.br/>. Último Acesso: 12/04/2016;

Java™ Platform, SE7 API Specification. Disponível em:

<https://docs.oracle.com/javase/7/docs/api/>. Último Acesso: 12/04/2016;

#### 5. Anexos

Imagens dos testes



```
entrada.txt x saida.txt x
E;11:03;KXK-3308;NE N1NE1
E;16:45;KYS-4520;VP N1VP1
E;04:35;DXJ-7636;VP Horário de Funcionamento: 06:00 - 21:00
E;15:12;ZVS-6781;MT N1MT1
E;10:36;JGN-4502;VG N1VG1
S;17:29;JGN-4502;MT MT;06:53;55,07
E;12:60;FSZ-6470;MT N1MT2
E;21:36;IVT-8279;MT N1VP2
E;18:56;ARI-8602;VG N1VG1
E;21:54;VUC-0735;VG N1VG2
E;07:60;KKB-5659;VG N2VG1
S;11:06;KKB-5659;MT MT;03:06;24,80
E;12:38;AIF-4089;MT N1VP3
E;18:50;UTK-7132;MT N1VP4
E;05:29;NNB-4063;VG Horário de Funcionamento: 06:00 - 21:00
E;00:32;PEC-4433;VG Horário de Funcionamento: 06:00 - 21:00
E;17:02;FFB-9986;VG N2VG1
S;07:18;FFB-9986;MT MT;-10:16;-77,87
E;09:45;TPD-0907;MT N2MT1
E;19:28;KQP-9836;MT N2MT2
E;12:53;TQI-8088;NE N1NE2
E;10:14;HDG-7021;MT N2VP1
E;05:38;OOU-6178;VG Horário de Funcionamento: 06:00 - 21:00
S;13:26;OOU-6178;MT Carro inválido!
```

Figura 3: Teste de horário de funcionamento e carro inválido

```
entrada.txt x  saída.txt x
E;21:46;OFB-7655;MT N1MT1
E;21:02;GJO-4731;VG N1VG1
E;11:25;PYT-5886;VP N1VP1
E;16:26;FNQ-2794;VP N1VP2
E;22:01;GUK-8035;VG N1VG2
E;14:41;CJG-2236;VG N2VG1
E;04:04;AHB-3434;VP N1VP3
E;01:36;EAE-9074;NE N1NE1
E;02:11;SDZ-5903;VG N2VG2
E;07:27;NQV-7306;VG N3VG1
E;08:58;JEC-6460;VP N1VP4
E;19:57;IMZ-0832;VP N2VP1
E;17:50;SUY-6660;VG N3VG2
E;08:48;EXW-1251;NE N1NE2
E;11:25;JLR-8546;MT N1MT2
E;12:15;OVS-3590;VG N4VG1
E;18:53;ULX-7236;VG N4VG2
E;21:08;YQF-3260;VP N2VP2
E;01:16;RGA-7147;NE N2NE1
E;19:51;SXH-2100;NE N2NE2
```

Figura 4: Teste de prioridade de estacionamento

```
entrada.txt x  saída.txt x
E;15:56;LYN-1863;NE N1VP1
E;15:03;MYB-7481;VG N1MT1
E;08:30;JVD-0290;VG N1VP2
E;18:55;TWX-2699;NE Horário de Funcionamento: 06:00 - 21:00
E;10:58;GHW-1187;VP Horário de Funcionamento: 06:00 - 21:00
E;13:11;HBZ-7562;MT N1VP3
E;19:00;PJI-3535;MT Horário de Funcionamento: 06:00 - 21:00
E;17:54;TWE-9055;MT Horário de Funcionamento: 06:00 - 21:00
E;21:05;OCI-0153;NE N1VP4
E;01:18;VCS-5662;VG N1VG1
E;05:23;FKY-4855;NE N1MT2
E;17:19;HQF-4323;NE N1VG2
E;12:10;LOQ-5661;NE N2MT1
E;04:18;AFD-9350;MT N2VG1
E;07:28;ZGU-8753;MT N2VP1
E;07:07;IAE-3119;NE Horário de Funcionamento: 06:00 - 21:00
E;16:01;PMW-5088;VG N1NE1
E;23:16;AJS-1838;VP N2MT2
E;09:14;BFI-8376;VG N1NE2
E;11:09;NIZ-5848;VG N2VP2
E;06:56;FRZ-8856;NE N2NE1
E;18:37;IUO-6102;NE N2VP3
E;03:59;LWS-5809;VG Horário de Funcionamento: 06:00 - 21:00
E;00:51;RSH-8917;NE N2VP4
E;07:24;HBO-5032;MT N2NE2
E;06:54;TVM-1149;VP N2VP2
E;03:40;JLB-2435;VG N3VG1
E;10:22;LUZ-3629;NE Horário de Funcionamento: 06:00 - 21:00
E;09:10;UZA-6975;VG N3NE1
E;06:47;PUO-6449;VP N3VP1
E;05:53;KYY-7010;NE N3VP2
E;06:35;JIM-2053;VP Horário de Funcionamento: 06:00 - 21:00
E;09:23;UUR-9288;VG N3VG2
E;04:53;OLI-1305;NE N3VP3
E;12:35;CWV-2508;NE Horário de Funcionamento: 06:00 - 21:00
E;14:42;XHC-7294;VG Horário de Funcionamento: 06:00 - 21:00
E;03:25;ZAI-9898;NE Horário de Funcionamento: 06:00 - 21:00
E;21:40;RIY-7531;MT Horário de Funcionamento: 06:00 - 21:00
E;23:12;SXW-4642;MT Horário de Funcionamento: 06:00 - 21:00
E;00:44;AFF-2542;NE N3NE2
E;00:34;VPB-4198;VG N4VG1
E;21:58;EUS-5299;VP N3MT1
E;14:22;ZLN-0146;MT N3VP4
E;22:11;SHK-2584;VP N4NE1
E;00:45;CIG-1412;NE N4VG2
E;09:27;HKE-8279;VG LOTADO!
E;17:28;TWH-2890;VP Horário de Funcionamento: 06:00 - 21:00
E;17:28;NAB-4331;VP Horário de Funcionamento: 06:00 - 21:00
E;09:27;PBC-2379;VG N4VP1
```

Figura 5: Teste de lotação