Exercícios de preparação para o 1º teste

1. Cofre com combinação – Classe CombinationLock

Um objecto desta classe tem um mostrador com 26 posições possíveis (de A a Z). O mostrador tem de ser fixado três vezes. Se for utilizada a ordem correcta, o cofre é aberto. Quando se fecha o cofre, é necessário voltar a introduzir a combinação. Se houver mais de 3 fixações seguidas, são as três últimas que contam. Apresente a lista das operações que a classe CombinationLock deverá implementar e defina a memória necessária a cada objecto da classe.

2. Classe CarDisplay

Definir a classe CarDisplay que representa o painel de ferramentas de um carro no que diz respeito aos quilómetros percorridos e ao seu gasto de combustível. Os objectos desta classe ajudam o condutor a saber:

- Qual a distância total percorrida pelo carro;
- Qual o consumo do carro por cada 100 Km, após o último abastecimento de combustível;
- Qual a distância percorrida desde o último abastecimento de combustível.

Queremos criar objectos das seguintes formas:

- Informando a quantidade em litros do tanque de combustível: neste caso o objecto é criado com a quilometragem a zero;
- Informando a quantidade em litros do tanque de combustível e a quilometragem inicial do carro.

Para simplificar o problema, assuma que só se abastece o carro quando o tanque fica na reserva. O valor da reserva do tanque é dado por 25% do total de combustível no tanque.

Os objectos da classe CarDisplay deverão poder executar as seguintes operações:

- **`float`** `addFuel()` : enche o tanque de combustível, indica o consumo do carro por cada 100 Km, com base nos quilómetros percorridos desde o último abastecimento de combustível e inicia um novo registo de consumo;
- **`float`** `addFuel(int ammount)` : adiciona a quantidade indicada de combustível ao tanque, indica o consumo do carro por cada 100 Km, com base nos quilómetros percorridos desde o último abastecimento de combustível e inicia um novo registo de consumo;
- **`int`** `getTotalKm`() : indica o número total de quilómetros percorridos pelo carro (quilometragem do carro)
- `int getCurrentKm`() : indica o número de quilómetros percorridos pelo carro (quilometragem do carro) desde o último abastecimento de combustível;

- **`void`** `addDistance(int distance)` : adiciona o número de quilómetros fornecidos como parâmetro, à quilometragem do carro.

Implemente a classe.

3. Classe MyMasterMind

Implemente uma classe para jogar o tradicional jogo de MasterMind, mas em vez de usar cores usa um número inteiro de cinco dígitos.

Quando o jogo é criado gera-se um número aleatório de cinco dígitos.
Os objectos da classe MyMasterMind deverão poder executar as seguintes operações:

- **`String`** `tryIt(int guess)` : este método recebe um inteiro com uma tentativa de acertar no número secreto e devolve uma String indicando:
  - O número de dígitos que se encontram na posição correcta. Por exemplo, se o número de dígitos correctos for 3, deve ser "3 match";
  - "You win!", caso o jogador tenha adivinhado o número.
- **`void`** `newGame()` : inicializa outro jogo (gerando um novo número aleatório).

No método `tryIt()`, se `guess` tiver menos de cinco digitos, assuma que o número dado tem zeros à esquerda.

4. Parque de estacionamento:

Implemente em Java a classe ParkingMeter, cujos objectos representam parquimetros.

Consoante o tempo de permanência no parque, assim é a tarifa aplicada, sendo as possíveis tarifas as seguintes:

- três primeiras horas: 1,00€/hora
- da 4ª hora à 7ª hora: 1,25€/hora
- horas restantes: 1,50€/hora.

O parquimetro determina que o máximo a pagar por dia é de 15,00€.

Considera-se também que o tempo de permanência não excede as 24h, mas a contagem pode começar num dia e terminar no dia seguinte.

Para o pagamento, é inserida uma quantia, e deverá ser possível calcular o troco a devolver.

As operações reconhecidas são as seguintes:

```
void restartCounting(int beginHour, int beginMinute);
```

Regista um novo inicio de contagem de tempo de permanência no parquimetro.

```
void stopCounting(int endHour, int endMinute);
```

Regista o fim de contagem do tempo de permanência no parquimetro.

```
double calculateFee();
```
Calcula o preço a pagar do último registo de permanência do parquimetro.

```
double calculateChange(double amountInserted);
```
Calcula o troco a devolver ao utilizador, caso seja feito um pagamento do último registo de permanência com a quantia dada como parâmetro. No caso de não ser possível efectuar o pagamento (a quantia dada não é suficiente), a operação devolve -1.

```
double calculateStayedTime();
```
Calcula o tempo total de permanência do último registo realizado no parquimetro. O resultado é devolvido em horas (p.e. se permanecer 1h30, o resultado devolvido será de 1,5).

5. Computador de bordo:

Cada objecto da class CarComputer representa um computador de bordo que permite controlar vários aspectos relacionados com o seu funcionamento, tais como a quilometragem, o consumo médio, entre outros.

Quando um CarComputer é criado, é dada a quantidade de litros de combustivel actuais. Se nada for indicado, então assume-se que o depósito se encontra cheio.

A capacidade do depósito de combustivel é de 60 litros, e a média de consumo é de 6 litros por cada 100km.

Para efectuar este controlo são disponibilizados os seguintes métodos:

```
double getGasLeft();
```
Devolve o combustivel restante no depósito.

```
void fillTank(double liters);
```
Enche o depósito com o número de litros de combustivel indicado. O limite do depósito não deve ser excedido.

```
double fullFill();
```
Enche o depósito até ao máximo permitido, e indica quantos litros usou.

```
double kmLeft();
```
Calcula o número de quilómetros que é possível efectuar com o combustivel actual.

```
double gasUsed(double distance);
```
Calcula o combustivel necessário para percorrer a distância pretendida.

```
boolean enoughGas(double numberOfKm);
```
Determina se é possível percorrer a distância pretendida com o combustivel actual.

```
double travel(double distance);
```

Viaja a distância pretendida, gastando assim combustivel. Caso não haja combustivel suficiente para toda a viagem, desloca-se até o depósito ficar vazio. Devolve a distancia efectivamente percorida.

A classe está definida do seguinte modo:

```
class CarComputer {
    static final double CAPACITY = 60.0;
    static final double AVERAGE_USE = 6.0;

    double gasLeft;

    CarComputer(double gas) {
        gasLeft = gas;
    }

    CarComputer() {
        gasLeft = CAPACITY;
    }

    double getGasLeft() {
        return gasLeft;
    }

    void fillTank(double liters) {
        if ((gasLeft + liters) > CAPACITY)
            gasLeft = CAPACITY;
        else
            gasLeft = gasLeft + liters;
    }

    double fullFill() {
        double beforeGas = gasLeft;
        gasLeft = CAPACITY;
        return CAPACITY - beforeGas;
    }

    double kmLeft() {
        return ((gasLeft * 100) / AVERAGE_USE);
    }

    double gasUsed(double distance) {
        return ((distance * AVERAGE_USE) / 100);
    }

    boolean enoughGas(double numberOfKm) {
        return this.kmLeft() >= numberOfKm;
    }
    double travel(double distance) {
        if (this.enoughGas(distance)) {
            gasLeft = gasLeft - this.gasUsed(distance);
            return distance;
        } else {
            gasLeft = 0;
            return this.kmLeft();
        }
    }
}
```

a) No teste seguinte, indique o valor retornado por cada método invocado:

```
CarComputer c1 = new CarComputer(10);
CarComputer c2 = new CarComputer(30);
CarComputer c3 = new CarComputer();
c2.kmLeft()
c1.enoughGas(170)
c3.travel(500.0)
c3.gasUsed(500.0)
c1.fullFill()
c2.fillTank(70);
c3.fullFill()
c2.kmLeft()
c3.kmLeft()
```