

O exercício foi desenvolvido para ser resolvido por módulos. Os módulos estão ordenados para serem resolvidos progressivamente e são dependentes, ou seja, precisam ser resolvidos em ordem.

**Arquivos a entregar:**

* **Playground SWIFT:**
  + Cada classe em um arquivo individual.
  + Caso de execução implementado.
* **Um diagrama UML que contenha todos os relacionamentos apresentados no exercício.**Não é necessário declarar os construtores, getters e setters em um diagrama UML.  
  Pode ser resolvido em papel com a entrega de uma foto ou utilizar uma ferramenta que exporte uma imagem.

**Aprovação do entregável:**

* **Todas** as tarefas devem ser realizadas corretamente.
* O exercício é INDIVIDUAL. Para tanto, **deve ser feita uma defesa oral do trabalho.**A defesa oral é basicamente para avaliar se o exercício foi realizado pelo aluno e confirmar a compreensão da matéria.

**Destaque:**

* Para obter uma qualificação de DESTACADO, deverão resolver corretamente não só este exercício, como também o que se encontra na seção adicional.

**Data limite para entrega:**

* Uma semana após entregue a lição, antes de começar a aula.
* Quem re-entregar o exercício, não poderá receber a qualificação DESTACADO.

**Parte A**

1. Criar um diagrama de classes que modele um objeto Aluno.   
   A princípio, um aluno possui um nome **(String**), um sobrenome **(String)** e um código de aluno **(Integer)**. Além disso, um aluno pode possuir um e-mail de contato **(String)**.
2. Implementar a classe criando os atributos necessários.
3. Criar o construtor, os getter e setter necessários para os atributos anteriores.

**Parte B**

1. Criar um diagrama de classes que modele um objeto Curso.   
   A princípio, um curso possui um nome **(String)** e um código de curso **(Integer)**.
2. Implementar a classe criando os atributos necessários.
3. Criar o construtor, os getter e setter necessários para os atributos anteriores.

**Parte C**

1. Realizar um diagrama de classe que modele um objeto Professor.   
   A princípio, um professor possui um nome **(String)**, um sobrenome **(String)**, um tempo de casa **(Int)** e um código de professor **(Int)**.
2. Implementar a classe criando os atributos necessários.
3. Criar o construtor, os getter e setter necessários para os atributos anteriores.

**Parte D**

Queremos adicionar duas categorias de professores ao modelo anterior.

Os professores titulares e os professores adjuntos. Um professor titular tem uma especialidade **(String)** e um professor adjunto tem uma quantidade de horas de monitoria **(Int)**.

1. Como você modificaria o diagrama de classe de Professor criado anteriormente?
2. Modificar a implementação considerando as novas alterações. Criar as classes necessárias e seus construtores correspondentes.
3. Criar os getters e setters necessários para os novos atributos.

**Parte E**

Além de ter um nome e código de curso, um curso possui um professor titular, um professor adjunto, uma quantidade máxima de alunos **(Int)** e uma lista de alunos matriculados. Tanto o professor titular como o adjunto podem não estar designados ao curso no momento de registrá-lo.

1. Como você modificaria o diagrama de classe de Curso criado anteriormente?
2. Modificar a implementação considerando as novas alterações.
3. Criar os getters e setters necessários para os novos atributos.

**Parte F**

1. Realizar um diagrama de classe que modele um objeto Matrícula.  
   A princípio, uma matrícula tem um aluno, um curso e uma data de matrícula **(Date).** Além disso, uma matrícula pode incluir informações sobre a forma de pagamento **(String)**.
2. Implementar a classe criando os atributos necessários.
3. Criar um construtor de Matrícula que tome como parâmetros um aluno e um curso e utilize a data atual como data de matrícula. A classe Date permite modelar datas em Swift e instanciar um objeto de tipo Date sem parâmetros. O objeto representa a data atual.  
   ***var DataAtual: Date = Date()***

**Parte G**

1. Criar um diagrama de classes que modele um objeto Manager.  
   A princípio, o Manager possui uma lista de alunos, uma lista de professores, uma lista de cursos e uma lista de matrículas.
2. Implementar a classe criando os atributos necessários.
3. Criar um construtor de Manager que não receba parâmetros.

**Parte H**

1. Criar um método na classe **Curso** que permita adicionar um aluno a uma lista de alunos do curso. O método retornará **true** se o aluno pôde ser adicionado ou **false** caso não haja disponibilidade de vaga.

**public func adicionarUmAluno( umAluno: Aluno ) -> Bool**

1. Criar um método na classe **Curso** que permita excluir um aluno da lista de alunos do curso.

**public func eliminarAluno( umAluno: Aluno ) -> Void**

1. Criar um método na classe **Curso** que permita designar um professor titular e um professor adjunto.

**public func designarProfessores( umTitular: ProfessorTitular, umAdjunto: ProfessorAdjunto ) -> Void**

**Parte I**

1. Criar um método na classe **Manager** que permita registrar um curso. O método recebe como parâmetros o nome do curso, o código e a quantidade máxima de alunos admitidos. O método deve criar um curso com os dados correspondentes e adicioná-lo à lista de cursos.

**public func registroCurso( nomeCurso: String, codigoCurso: Int, numeroMaximoDealunos: Int ) -> Void**

1. Criar um método na classe **Manager** que permita eliminar um curso. O método recebe como parâmetro o código do curso. O método deve utilizar o código do curso para encontrá-lo na lista de cursos e excluí-lo da lista.

**public func eliminarCurso( codigoCurso: Int ) -> Void**

1. Criar um método na classe **DigitalHouseManager** que permita registrar um professor adjunto. O método recebe como parâmetros o nome do professor, o sobrenome, o código e a quantidade de horas disponíveis para monitoria. O tempo de casa inicial do professor será zero. O método deve criar um professor adjunto com os dados correspondentes e adicioná-lo à lista de professores.

**public func registrarProfessorAdjunto( nome: String, sobrenome: String, codigoProfessor: Int, quantidadeDeHoras: Int ) -> Void**

1. Criar um método na classe **Manager** que permita registrar um professor titular. O método recebe como parâmetros o nome do professor, o sobrenome, o código e a especialidade. O tempo de casa inicial do professor será zero. O método deve criar um professor titular com os dados correspondentes e adicioná-lo à lista de professores.

**public func registroProfessorTitular( nome: String, sobrenome: String, codigoProfessor: Int, especialidade: String ) -> Void**

1. Criar um método na classe **Manager** que permita eliminar um professor. O método recebe como parâmetro o código do professor. O método deve utilizar o código do professor para encontrá-lo na lista de professores e eliminá-lo da lista.

**public func eliminarProfessor( codigoProfessor: Int ) -> Void**

1. Criar um método na classe **Manager** que permita registrar um aluno. O método recebe como parâmetros o nome, o sobrenome e o código do aluno. O método deve criar um aluno com os dados correspondentes e adicioná-lo à lista de alunos.

**public func registroAluno( nome: String, sobrenome: String, codigoAluno: Int ) -> Void**

1. Criar um método na classe **Manager** que permita matricular um aluno em um curso. O método recebe como parâmetros o código do aluno e o código do curso em que ele está se matriculando.

**public func matricularAluno( codigoAluno: Int, codigoCurso: Int ) -> Void**

O método deve:

* Encontrar o curso em que o aluno está se matriculando.
* Encontrar o aluno que queremos matricular.
* Tentar adicionar o aluno ao curso, usando os métodos da classe Curso.
* Caso seja possível:
  + Criar uma matrícula e configurá-la com os dados correspondentes.
  + Adicionar a matrícula à lista de matrículas.
  + Informar na tela que a matrícula foi realizada.
* Se não for possível inscrever o aluno (por exemplo, se não houver vaga):
  + Informar na tela que não foi possível realizar a matrícula porque não há vagas.

1. Criar um método na classe **Manager** que permita Em seguida, explicar como ‘designar’ uma classe a uma cena no storyboard. professores a um curso. O método recebe como parâmetros o código do curso, o código do professor titular e o código do professor adjunto.

**public func designarProfessores( codigoCurso: Int, codigoProfessorTitular: Int, codigoProfessorAdjunto: Int ) -> Void**

O método deve:

* Encontrar o professor titular na lista de professores.
* Encontrar o professor adjunto na lista de professores.
* Designar ambos professores ao curso.
* Se não for possível designar algum dos professores (por exemplo, porque o código não existe), não se deve designar nenhum dos dois.

**Caso de Exemplo (Parte J)**

Este é um caso de exemplo que simula o uso do sistema num entorno real, usando o Playground e as classes criadas.

1. Instanciar um Manager.

1. Registrar dois professores titulares e dois professores adjuntos. (Inventar todos os seus valores)
2. Registrar dois cursos.  
   * **Nome do curso: iOS   
     Código do curso: 20001  
     Quantidade máxima: 3**
   * **Nome do curso: Android   
     Código do curso: 20002  
     Quantidade máxima: 2**
3. Designar um professor titular e um adjunto para cada curso.
4. Registrar quatro alunos. (Inventar todos os seus valores).
5. Matricular dois alunos no curso de iOS.
6. Matricular dois alunos no curso de Android.
7. Matricular dois outros alunos no curso de iOS.
8. Tentar matricular um aluno em um curso inexistente.
9. Tentar matricular um aluno inexistente a um curso.