Documentação do Projeto: iSoccer

- Funcionalidades: O sistema possui todos os requisitos funcionais solicitados na descrição do Projeto. Seu objetivo principal é fornecer uma excelente administração dos recursos de um time de futebol. Ao iniciar o Sistema, deve-se fornecer um login e uma senha de administrador do mesmo, para que seja possível executar as demais ações. O programa possui diversas funcionalidades, desde adição de recursos físicos(ônibus, estádios e centros de treinamento) a adição de profissionais que compõe o time, tais como médicos, jogadores, técnicos, advogados e etc. Também é possível fazer alterações nestes recursos, algumas delas são: alteração em status de jogadores e status de sócios e . alterações em características dos estádios. A descrição das funcionalidades mais interessantes(não comentarei a maioria dos adds) é feita em nível de desenvolvedor abaixo:
    1. Autenticação de Administrador - ao executar o programa um login e senha são adicionados em um ArrayList através do método Add_Admin, agora cabe ao usuário digitar o login e senha correspondente e confirmar sua autenticidade. A autenticação é feita a partir de uma única checagem, que compara os dados fornecidos com os presentes no Array.
    2. Alteração do Status de um Sócio e atualização do valor da Contribuição - ao selecionar esta opção o usuário é obrigado a fornecer o nome e o cpf do sócio que deseja administrar, em seguida, o sistema faz uma busca através do método Search_Partner que funciona de uma forma bem simples: ao fornecer o nome e o cpf, uma busca é feita no ArrayList em que todos os sócios estão adicionados, com o auxílio de um for quando o sócio é encontrado o valor booleano da variável found é alterado de “false” para “true” e o método retorna o número 1 como sinal que a busca foi realizada com sucesso, caso o sócio não tenha sido encontrado 0 é retornado e o usuário é informado que aquele sócio não existe. Para mudar o valor da contribuição o passo inicial é bem semelhante, basta digitar um novo valor e o método Alter_Contribution se encarrega do resto, basicamente efetuando uma busca com o cpf do sócio no ArrayList e substituindo o campo: valor da contribuição, pelo novo valor digitado.
    3. Administração de Recursos - sem dúvidas a funcionalidade mais complexa, é nesta funcionalidade onde todos os demais recursos são adicionados ou checados. O sistema pede que você digite(de acordo com um número digitado) qual recurso você deseja gerenciar. Ônibus podem ser adicionados ou checados em disponibilidade, para checar a disponibilidade o programa usa o método Check_Status(da classe Resources da qual Bus herda) e funciona de forma muito similar a Search_Partner citada acima, ao efetuar a busca com o auxílio de um for dentro o ArrayList de Resources ele retorna Available caso haja algum Ônibus no time e Not Available caso não exista. A parte de checagem dos demais recursos é análoga. Ademais, os Estádios têm

opções a mais como quantidade de banheiros ou quantidade de lanchonetes, que também funcionam de maneira análoga às demais checagens.

4. Gerar Relatórios - aqui é onde as informações são mostradas de acordo com o que o usuário quiser ver. Entretanto, todas as informações são mostradas na tela de forma similar a todas as outras: um for é iniciado desde o início de cada ArrayList até o seu fim, e para cada posição são printadas todas as informações presentes naquele Array, de nome a afins.

- Classes: O objetivo foi criar classes com os mínimos atributos necessários e mais comuns, dessa forma eu poderia usar um conceito muito conhecido em OO: Herança. A partir deste objetivo, foi criada a classe Worker com atributos mais comuns(nome, email, cpf, salário e telefone) e outras 7 classes herdaram seus atributos e receberam alguns próprios(a explicação com a classe Resource para representar ônibus, centro de treinamento e estádio é análoga). As vantagens foram incríveis: Economia enorme de tempo(pois não preciso reescrever os atributos comuns, uma vez que eles são herdados), melhor organização do código e por fim melhor uso de armazenamento(exclusivamente em Resource). Não enxerguei desvantagens no processo de criação de classes.
- Distribuição de Métodos: Nesta parte eu quis organizar os métodos e colocá-los em suas determinadas Classes, como por exemplo, o método Add_Medical é um método exclusivo da classe Medical, com isso eu pensei conseguir melhor organização do meu código como um todo. Entretanto, foi muito trabalhoso escrever cada método de cada classe de forma exclusiva(no que diz respeito aos de adicionar), por isso comecei a generalizar mais a partir de um certo ponto e alocar alguns métodos de uma forma mais usual, como por exemplo, os métodos Check_Status e Check_Quantity(que pertencem a classe Resource) e podem ser usados por todos os seus filhos. Uma desvantagem nessa forma de distribuição de métodos é que eu poderia ter generalizado mais as coisas(como por exemplo um método de adicionar funcionário, que adiciona todos os funcionários independente de ser um Médico, ou Advogado ou etc), acredito que isso seria bem mais simples e deixaria tudo mais sofisticado.
- Herança: O principal motivo para o uso de herança foi a brilhante possibilidade de não precisar reescrever atributos comuns a mais de 1 classe(neste caso 7 só para funcionário e seus tipos). A herança está presente em boa parte do código, possibilitando escrever bem menos código e generalizando atributos como: nome, email, cpf, telefone e salário. Acredito que a principal vantagem foi a economia dos dados, criando uma classe Pai eu poderia fazer com que seus filhos herdem todos os seus atributos, o que foi a peça chave na utilização de herança. Não vejo desvantagens no uso deste conceito.
- Classe Abstrata: A minha principal motivação no uso deste conceito foi: Tenho vários tipos de funcionários, mas não tenho um funcionário. O que eu quero dizer é que eu tenho muitos tipos de trabalhadores(jogadores, treinadores, médicos e etc), mas nem um deles vai ser um trabalhador somente, logo eu não preciso instanciá-lo(nem

devo). A maior vantagem foi proibir que a classe Woker fosse instanciada e não só isso. Também não enxergo desvantagens neste processo.

- Reuso: A utilização de reuso se deu por um único motivo: não precisar reinventar a roda toda hora. Com isto quero dar ênfase a um método muito importante chamado Check_Status, esse método foi utilizado para checar determinado status no programa e foi usado em várias ocasiões, bem como em classes diferentes. Não enxergo desvantagem alguma no uso deste recurso.
- Extensibilidade: A motivação neste ponto, foi manter os atributos da classe Pai e com o uso de herança passá-los a seus filhos. Eu diria que a melhor vantagem foi não precisar reescrever todos os métodos e atributos que estendi. Não enxergo desvantagens no uso deste conceito.