Nome:

Turma: 3

Número:

Os únicos pontos que foram retirados, foram na questão 2.a, pois não mencionei uma Biblioteca.

1.a – Proponho a funcionalidade "Alertas de Carreira" gerados durante a simulação, para fazer o aviso de aproximação do tempo máximo ou da idade-limite no posto. Esta funcionalidade implica criar uma hierarquia (ABC + alertas concretos) para cumprir OCP, o que irá permitir acrescentar novos tipos de alerta (por tempo no posto, idade, mérito, etc.) sem alterar o motor de simulação (GestorCarreiras) nem os relatórios existentes.

1.b – Para implementar esta nova funcionalidade tenho de criar AlertaCarreira(ABC) num novo ficheiro alertas.py. A classe concreta que desta deriva é AlertaTempoNoPosto(SerializavelMixin, AlertaCarreira) também em alertas.py.

1.c – Utilizarão os dados de militar.py (Militar.tempo_no_posto() e Militar.posto_atual) e regras_carreira.py (tempo máximo/idade). São gerados no ciclo de simulação (SimuladorCarreira) e exportados como os relatórios (via SerializavelMixin).

1.d – A funcionalidade transversal é a exportação JSON, que já existia para relatórios através de SerializavelMixin.exportar_json() em mixins.py e é reutilizada agora também na hierarquia de alertas (sem duplicação).

1.e – Reutilizei o mixin existente SerializavelMixin (mixins.py) para exportar JSON, e apliquei a uma classe de relatórios já existente (RelatorioIndividual em relatorio_individual.py) e a uma classe da nova hierarquia (AlertaTempoNoPosto em alertas.py).

1.f –

MRO RelatorioIndividual: RelatorioIndividual → SerializavelMixin → ValidacaoRegistoMixin → RegistoBaseMixin → RelatorioSimulacao → RenderizavelTexto → ExportavelJSON → object

MRO AlertaTempoNoPosto: AlertaTempoNoPosto → SerializavelMixin → AlertaCarreira → ABC → object

Escolhi SerializavelMixin primeiro para garantir que exportar_json() é resolvido de forma direta e consistente, mantendo o restante comportamento cooperativo nos mixins de registo/validação.

2.a – Atualmente, o projeto apenas gera gráficos a partir de relatórios/eventos, pelo que proponho a introdução de uma funcionalidade que desenhe uma linha temporal da progressão individual na carreira de um militar, marcando as várias promoções no tempo.

2.b – Atualmente a lógica de carreira está no domínio/gestão e a visualização isolada em graficos.py. Se implementar esta funcionalidade fazendo Militar.gerar_grafico_timeline() ou chamando Matplotlib dentro de GestorCarreiras.processar_passo() para ir desenhando à medida que gera eventos, o que faz com que os objetos de domínio que abrem figuras e formatam eixos, os quais dependem de Matplotlib, o que constitui uma violação do Princípio da Responsabilidade Única (SRP), pois Militar deixa de ser apenas modelo de carreira e passa a acumular responsabilidades de apresentação, e qualquer alteração estética do gráfico obriga a mexer em classes de domínio.

2.c – Para implementar esta funcionalidade sem quebrar o SRP tinha de manter o domínio e a gestão inalterados, ou seja militar.py e gestor_carreiras.py. Para gerar a linha temporal utilizaria os dados já agregados em RelatorioIndividual (relatorio_individual.py), evitando acesso direto ao estado interno do Militar. Criaria ainda uma classe específica de visualização em graficos.py, ou um novo TimelineCarreira, que seria responsável por transformar o relatório num gráfico Matplotlib. A chamada a esta classe ocorreria apenas no controlador_simulacao.py ou no main.py, garantindo que a lógica de apresentação se mantenha separada da lógica de domínio/gestão, respeitando o SRP.

3.a – Em militar.py, a classe Militar tem fragilidade de encapsulamento/proteção de estado pois atributos públicos como na_reserva e data_reserva (alteráveis externamente), podem colocar o objeto num estado incoerente (ex.: na_reserva=True com data_reserva=None).

3.b – Tornando o estado de reserva privado e expondo apenas propriedades de leitura, centralizando a transição num método passar_a_reserva(quando), resolve o problema pois o único caminho de mudança passa a validar e atualizar o estado de forma consistente.

3.c – No início do semestre tendia a assumir que as estruturas de dados se manteriam estáveis ao longo do projeto. No entanto, à medida que o projeto evoluiu, tive de lidar com diferenças no formato dos eventos gerados pela simulação (por exemplo, o uso de data ou quando), o que obrigou a revisões nos relatórios e mixins. Esta situação fez-me perceber que decisões aparentemente pequenas podem propagar-se por várias classes quando não existem contratos bem definidos. Como consequência, passei a valorizar mais interfaces explícitas, adaptação centralizada e isolamento de responsabilidades, em vez de soluções rápidas distribuídas pelo código.