IMD0040 Trabalho 3

Introdução

Este trabalho foi projetado para demonstrar seu entendimento no material apresentado pelo capítulo 10. Além disso, sugerimos a leitura do capítulo 11 antes de começar a fazer o trabalho. Você vai entregar o trabalho eletronicamente via SIGAA. O trabalho pode ser realizado em duplas. Caso seja esse o caso, você deverá informar sua dupla por e-mail até uma semana antes do prazo de entrega. Aqueles que não se manifestarem dentro deste prazo serão considerados como fazendo o trabalho individualmente.

Este trabalho pode ser desenvolvido utilizando-se de quaisquer ferramentas de desenvolvimento (IDE) que você quiser.

Qualquer método escrito por você deverá ser totalmente comentado utilizando tags javadoc.

Prazo de entrega: 23:59 11 de Outubro de 2018

A Tarefa

Este trabalho consiste na implementação de uma simulação caça/caçador.

O cenário é de um ambiente marítimo, consistindo principalmente de água, mas poderá incluir outras áreas que não sejam água (terra, pedras, etc). Neste ambiente existem pelo menos os seguintes participantes: sardinhas (Sardine), atuns (Tuna), tubarões (Shark) e algas. Cada um deles apresenta um comportamento específico.

Seu ponto de partida é o conjunto de classes Java que foi disponibilizado com essas instruções. Elas contêm todo código necessário para produzir a interface gráfica, mas sem qualquer implementação para a lógica da aplicação.

Em particular: As classes `SimulatorView`, `OceanStats` e `Counter` estão completas. Você não precisa alterar essas classes (a não ser que você queira).

As classes `Simulator` e `Ocean` estão incompletas. Você terá que adicionar bastante código nestas classes. A classe `Ocean` contém alguns esqueletos de métodos. Não remova ou modifique as assinaturas desses métodos, pois eles são utilizados por outras classes. Você precisará escrever a implementação desses métodos, e adicionar quaisquer outros métodos que achar necessário.

A classe `Simulator` precisará criar objetos de todas as outras classes (`SimulatorView`, `Ocean`, e todas as criaturas da simulação). Esta classe deverá invocar os seguintes métodos da classe `SimulatorView` para visualizar o mundo:

void setColor(Class fishClass, Color color)
- Define a cor que será usada para exibir peixes dessa classe

void showStatus(int step, Ocean ocean)
- Mostra o oceano especificado como parâmetro no passo (step) passado como parâmetro

boolean isViable(Ocean ocean)
- Retorna false se a simulação deve parar de rodar. Isso acontecerá se existir somente um tipo de

peixe no mundo.

Você precisa adicionar, pelo menos, uma classe para cada tipo de peixe da sua simulação.

Sua implementação deverá atender aos seguintes requisitos:

- Precisa ser uma simulação contínua
- Precisa ter uma classe principal que contém uma coleção (ou coleções) de atores. Esta classe deverá conter um loop para realizar a iteração nos atores da coleção chamando o método act de cada ator (similar a simulação foxes-and-rabbits).
- A aplicação precisa produzir uma saída gráfica e animada (similar a simulação foxes-and-rabbits).
- Cada localização (célula no mundo) possui um valor de alga na faixa [0..10]. Alga se regenera regularmente, e se espalha, se não for consumida por um peixe. (Com a visualização disponibilizada você não conseguirá ver as algas. Isso não é um problema. Se você realmente quiser, você poderá fazer com que as algas apareçam visualmente – basta modificar a classe SimulatorView para isso).
- Sardinhas comem algas.
- Atum come sardinha.
- Tubarões comem sardinha ou atum, mas eles preferem atum.
- Tubarões são solitários – eles preferem não nadar próximos um dos outros.
- Todos os peixes se reproduzem; todos os peixes podem morrer de fome.

Requisitos de desafio

- Sardinhas exibem um comportamento de flocking, isto é, tendem a nadar juntos (com algumas exceções ocasionais), tendem a manter uma mesma direção (com algumas exceções). Se elas percebem um predador se aproximando, elas entram em pânico e se espalham.

Pontuação e Qualidade de código

Parte da nota (30%) será atribuída simplesmente por completar uma solução básica que compila e roda. A maioria da nota (40%) será baseada no design da sua solução. Seu código deverá ser documentado (10%). 20% da nota será baseada no requisito de desafio.

- Escreva código orientado a objeto. Pense sobre design baseado em responsabilidade, focando em baixo acoplamento e alta coesão.
- Pense bem onde colocar código (ex: classe pai ou sub-classe? Qual a relação entre as classes?). Evite duplicação ou que uma classe saiba muito sobre outra.
- Use os recursos apropriados da linguagem Java quando necessário (ex: pacotes, tipos genéricos, classes abstratas e interfaces, final e enumerações).
- Mantenha um estilo de código consistente.
- Documente suas classes e métodos usando javadoc (veja http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html)

Finalmente

Antes de submeter seu trabalho, teste exaustivamente o projeto inteiro para se certificar de que nada do que foi acrescentado recentemente quebrou algo adicionado anteriormente. Se você não conseguir completar o projeto - mesmo se você não conseguir compilar – submeta o que você tem, pois é provável que você vai ter pelo menos algum ponto por isso.