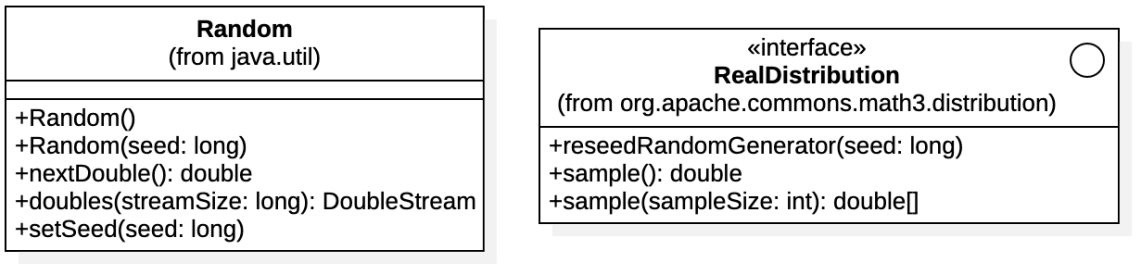
Geradores de Números Aleatórios (zip)

Em qualquer linguagem de programação existem geradores de números pseudo-aleatórios. Estes geradores são chamados assim pois os números não são de fato aleatórios, uma vez que é utilizada uma equação para calcular cada número gerado. Em Java, a classe `Random` do pacote `java.util` é uma das formas utilizadas para gerar tais números. No entanto, existem diferentes implementações destes geradores, cada uma com finalidades específicas ou diferentes níveis de qualidades em relação à aleatoriedade dos números gerados.

Bibliotecas como a Apache Commons Math fornecem, dentre outras funcionalidades, diversos geradores de números pseudo-aleatórios.

1. O problema

Existe uma grande variedade de implementações de tais geradores, cada implementação fornecendo uma interface diferente da outra. Para este projeto, observe o diagrama de classes abaixo.



Ele apresenta alguns métodos da classe `Random` do pacote `java.util` e da interface `RealDistribution` da biblioteca Apache Commons Math. Algumas das classes que implementam esta última interface são `NormalDistribution` e `LogNormalDistribution`. Como podem ver, existem diferenças entre as classes/interfaces, tornando também diferente a forma de utilizar cada uma das implementações disponíveis.

A seguir são apresentados os detalhes das diferentes implementações.

1.1. Classe Random da plataforma Java

Na classe `Random` temos os métodos `nextDouble()` e `doubles(long streamSize)`. Os dois podem ser utilizados para gerar números aleatórios. O primeiro gera um único número a cada vez que for chamado. O segundo gera um conjunto de números, cuja quantidade é definida pelo parâmetro `streamSize`. Assim, chamando `doubles(3)`, serão gerados 3 números aleatórios.

Adicionalmente, esta classe possui 2 construtores: um que não recebe parâmetro algum e outro que recebe uma `seed`. A `seed` (semente) é apenas um número que define o valor inicial do gerador. Se instanciarmos um gerador com a mesma `seed`, a sequência de números gerados será sempre a mesma. Se alterarmos a `seed` ou não informarmos uma, a cada vez que executarmos a aplicação, a sequência de números gerados será diferente.

1.2. Biblioteca Apache Commons Math

Já nas classes que implementam a interface `RealDistribution` da biblioteca Apache Commons Math (como `NormalDistribution` e `LogNormalDistribution`), os métodos que retornam um número aleatório ou um conjunto de números são diferentes. Temos os métodos `sample()` e `sample(int sampleSize)`, semelhantes aos métodos da classe `Random`, mas com nomes, tipos de parâmetro e retorno diferentes.

Por fim, nestas classes, se for preciso definir uma seed, isto não é feito por meio de um construtor, mas chamando o método `reseedRandomGenerator(long seed)`.

2. O Desafio

Identifique qual padrão de projeto pode ser aplicado para uniformizar a interface das classes `Random` do pacote `java.util` e das classes que implementam a interface `RealDistribution` da biblioteca Apache Commons Math. O padrão deve então permitir que a forma de utilizar qualquer destas classes seja a mesma.

Altere a aplicação de exemplo nesta pasta (que já inclui a dependência para a biblioteca Apache Commons Math no arquivo pom.xml) para aplicar o padrão de projeto adequado. Por fim, modifique a classe `Principal` para utilizar as classes criadas para o padrão.

3. Detalhes de Implementação

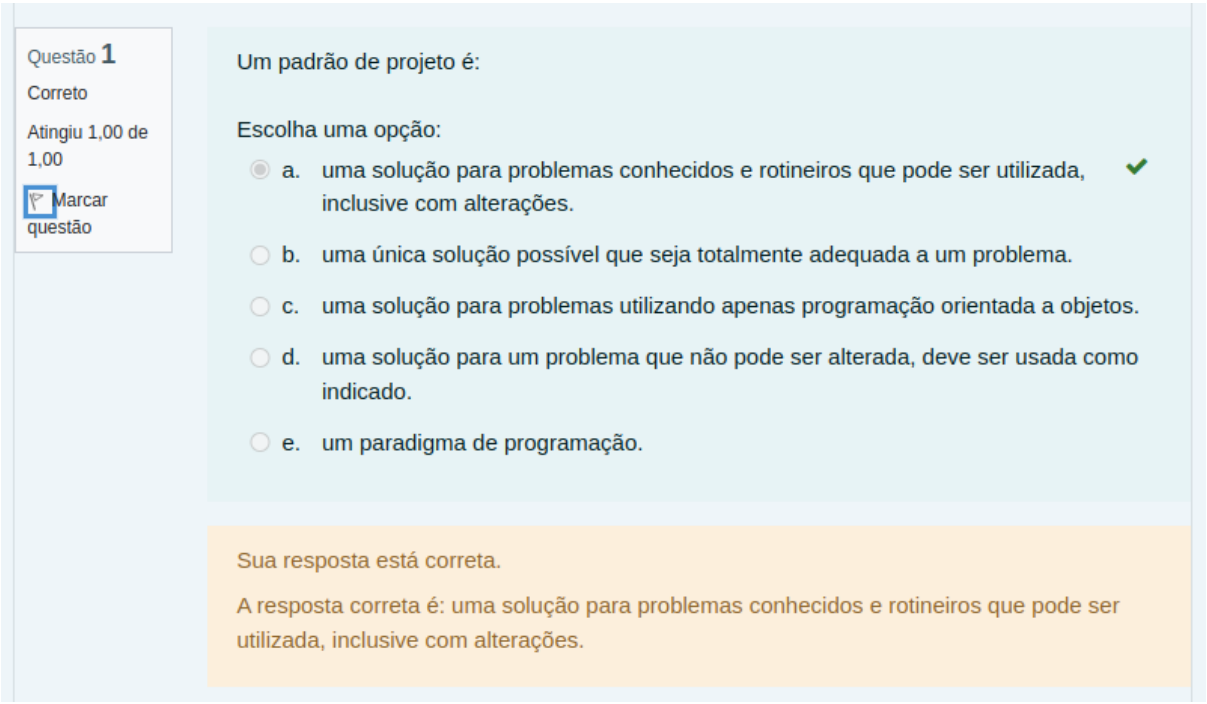
Observe que os métodos `sample(int sampleSize)` da interface `RealDistribution` e `doubles(long streamSize)` da classe `Random` tem tipos de parâmetros e retorno diferentes. O primeiro retorna um vetor de double e o segundo retorna um objeto `DoubleStream`. Uma maneira de uniformizar tais métodos é fazer com que o segundo também retorne um vetor de double. O código para isto pode ser:

```
double[] numeros = random.doubles(10).toArray();
```

Neste exemplo, estamos chamando o método `doubles()`, indicando que desejamos gerar 10 números aleatórios e convertendo de `DoubleStream` para um vetor usando o método `toArray()`.

Estamos padronizando fazendo os métodos retornarem vetores, pelo fato destes serem mais familiares que Streams (como é o caso do DoubleStream). Streams são mais modernos, seguros e flexíveis que vetores, mas vamos utilizar apenas vetores para simplificar.

prova teorica:



Questão **2**

Correto

Atingiu 1,00 de 1,00

Marcar questão

O padrão de projeto Singleton tem por objetivo:

Escolha uma opção:

a. permitir que uma classe atenda ao Single Responsibility Principle

b. criar famílias de objetos por meio de uma única interface.

c. permitir a variação de interfaces.

d. assegurar que uma única classe no sistema tenha uma única instância.

e. assegurar que a classe a qual o padrão é aplicado tenha uma única instância. ✔

Sua resposta está correta.

A resposta correta é:
assegurar que a classe a qual o padrão é aplicado tenha uma única instância.

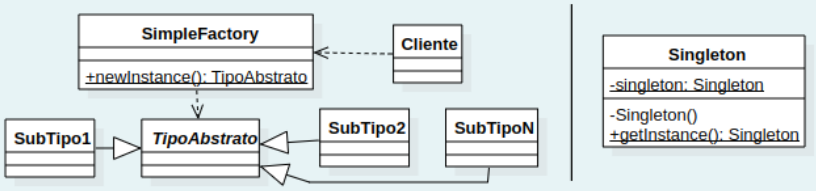
Questão **3**

Parcialmente correto

Atingiu 1,33 de 2,00

Marcar questão

Analise os diagramas do Simple Factory e Singleton abaixo:



Escolha uma ou mais:

a. Ao utilizar frameworks de injeção de dependência que fornecem recursos para definir uma classe como Singleton, é preciso ter cuidado na utilização do objeto criado pois há margem para erros que permitem a criação de objetos adicionais da classe. ✔

b. Se o método newInstance é estático, não faz sentido ter um construtor público. ✔

c. Para garantir que exista apenas uma instância de um Singleton, a única forma é definir o atributo *instance* como static final e inicializá-lo na declaração.

d. Uma das diferenças do SimpleFactory pro Singleton é que o primeiro pode criar uma única instância de diversas classes, enquanto o Singleton só pode criar uma única instância da classe que implementa o padrão.

e. Os dois são padrões criacionais. Um encapsula e centraliza a criação de diferentes classes. O outro se encarrega de criar um objeto de uma classe específica. Um precisa de um construtor privado. Analisando o diagrama do outro, também podemos fazer o mesmo.

Questão **4**

Incorreto

Atingiu 0,00 de 1,00

Marcar questão

Em relação aos padrões de projeto do GoF:

Escolha uma opção:

a. O padrão Factory Method permite que uma superclasse seja a única responsável por decidir qual objeto será instanciado em qualquer momento. ✖

b. O padrão Builder é atualmente desnecessário, uma vez que temos IDEs que geram implementações para isto.

c. Simple Factory é o padrão GoF mais simples para encapsular a criação de objetos, inclusive é o primeiro desta categoria que aprendemos.

d. O padrão Strategy atende ao princípio "Favorecer composição no lugar de Herança" pois flexibiliza a troca de um comportamento por outro.

e. O padrão Strategy atende ao princípio "Favorecer composição no lugar de Herança", mas podemos trocar um comportamento por outro em tempo de execução apenas usando herança. Então, aplicar ou não o padrão depende dos requisitos do sistema.

Sua resposta está incorreta.

A resposta correta é: O padrão Strategy atende ao princípio "Favorecer composição no lugar de Herança" pois flexibiliza a troca de um comportamento por outro.

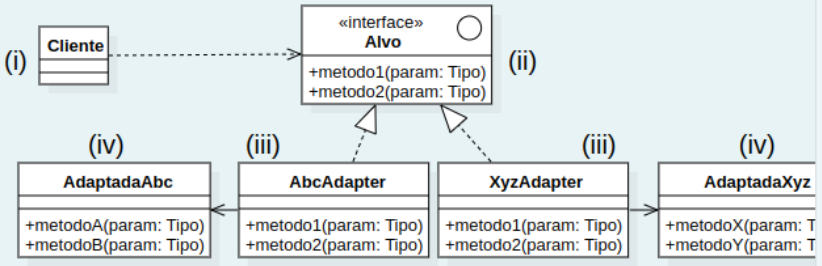
Questão **5**

Correto

Atingiu 2,00 de 2,00

Marcar questão

O diagrama abaixo apresenta a modelagem do padrão Adapter.



Escolha uma opção:

a. (i) usa diretamente as classes adaptadas; (ii) faz com que os adaptadores tenham estruturas diferentes; (iii) são classes; (iv) herdam indiretamente métodos da interface Alvo.

b. A modelagem apresentada está incorreta.

c. (i) é uma pessoa; (ii) deve ser implementada pelo cliente; (iii) são os adaptadores que fornecem uma interface comum pras classes adaptadas; (iv) são classes que você deve criar.

d. (i) depende da interface `Alvo`; (ii) define uma interface pública comum aos adaptadores; (iii) adaptadores que padronizam o uso das classes adaptadas; (iv) classes incompatíveis que tornam seu uso diferente uma da outra. ✔

e. (i) implementa `Alvo`; (ii) deve ser implementada pelas classes adaptadas; (iii)

Questão **6**

Parcialmente correto

Atingiu 0,33 de 1,00

Marcar questão

Marque as alternativas corretas

Escolha uma ou mais:

- [x] a. Padrões criacionais visam encapsular o processo de criação de objetos, escondendo o tipo concreto dos objetos a serem criados. ✔
- [ ] b. Template Method é um padrão estrutural, pois as subclasses podem alterar a assinatura do método.
- [ ] c. Padrões comportamentais estão relacionados à implementação de métodos que representam algoritmos ou partes de algoritmos.
- [ ] d. Padrões de projetos representam soluções para problemas conhecidos e podem ser aplicados em conjunto. Nestes casos, podem até mesmo sofrer alterações.
- [ ] e. Padrões estruturais visam alterar a estrutura de objetos depois de instanciados.

Sua resposta está parcialmente correta.

Você selecionou corretamente 1.
As respostas corretas são: Padrões criacionais visam encapsular o processo de criação de objetos, escondendo o tipo concreto dos objetos a serem criados., Padrões comportamentais estão relacionados à implementação de métodos que representam algoritmos ou partes de algoritmos., Padrões de projetos representam soluções para problemas conhecidos e podem ser aplicados em conjunto. Nestes casos, podem até mesmo sofrer alterações.

algoritmos ou partes de algoritmos., Padrões de projetos representam soluções para problemas conhecidos e podem ser aplicados em conjunto. Nestes casos, podem até mesmo sofrer alterações.

Questão **7**

Correto

Atingiu 1,00 de 1,00

Marcar questão

Qual opção abaixo representa padrões de projeto comportamentais?

Escolha uma opção:

- ( ) a. Useless, Useful e Util
- ( ) b. Singleton, Adapter e Strategy
- ( ) c. Builder e Singleton
- (•) d. Strategy, Template Method e outros ✔
- ( ) e. Adapter e Strategy

Sua resposta está correta.

A resposta correta é: Strategy, Template Method e outros

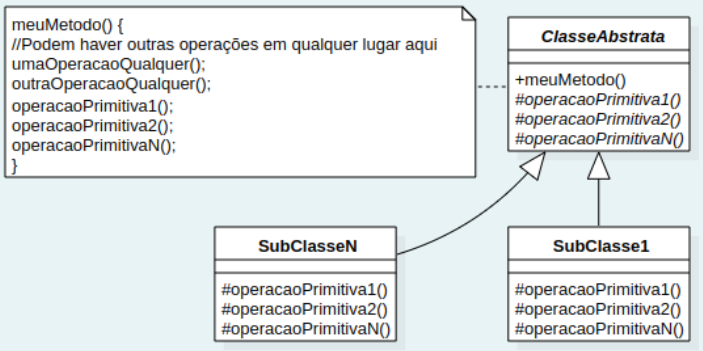
Questão **8**

Parcialmente correto

Atingiu 0,33 de 1,00

Marcar questão

Em relação ao diagrama abaixo de um padrão de projeto, marque as alternativas corretas.



Escolha uma ou mais:

- [x] a. Os métodos *operacaoPrimitiva* 1 a N representam parte do comportamento do *meuMetodo*, mas a implementação deles é definida apenas nas subclasses. ✔
- [ ] b. Como as operações primitivas estão sendo chamadas no *meuMetodo* na superclasse, isso significa que ela fornece implementações padrões de tais métodos.
- [ ] c. Um dos objetivos deste padrão é garantir que um conjunto de instruções seja sempre executado numa ordem definida pela superclasse, pois o funcionamento do algoritmo depende de tal ordem.
- [ ] d. O *meuMetodo* é implementado na superclasse e, neste padrão, não deve ter sua implementação alterada pelas subclasses.
- [ ] e. Pelo diagrama, percebemos que os métodos *operacaoPrimitiva* 1 a N podem ter uma implementação na superclasse, mas podem ser sobrescritos na subclasse.

Sua resposta está parcialmente correta.

Você selecionou corretamente 1.
As respostas corretas são: O *meuMetodo* é implementado na superclasse e, neste padrão, não deve ter sua implementação alterada pelas subclasses., Os métodos *operacaoPrimitiva* 1 a N representam parte do comportamento do *meuMetodo*, mas a implementação deles é definida apenas nas subclasses., Um dos objetivos deste padrão é garantir que um conjunto de instruções seja sempre executado numa ordem definida pela superclasse, pois o funcionamento do algoritmo depende de tal ordem.