Computação 2 — 2020/1 — Professor Vinícius Gusmão — Lista 1

Veja o código abaixo. Serão feitas perguntas sobre ele.
Assuma que cada classe ou interface está em um arquivo só dela, como de costume, e que todos os imports foram feitos corretamente, embora não apareçam aqui.

```java
public interface Sorteador {

    /** Retorna um inteiro aleatório. */
    int sortear();

    /** Retorna a média de todos os sorteios já feitos. */
    float getMediaSorteios();
}

public class Jogo {

    private Sorteador sorteador;

    public Jogo(Sorteador sorteador) {
        this.sorteador = sorteador;
    }

    public void jogar(int numeroDeRodadas) {
        int vitoriasJogadorA = 0;
        int vitoriasJogadorB = 0;
        int empates = 0;
        for (int i = 1; i<= numeroDeRodadas; i++) {
            int resultadoJogadorA = sorteador.sortear();
            int resultadoJogadorB = sorteador.sortear();
            if (resultadoJogadorA > resultadoJogadorB) {
                vitoriasJogadorA++;
            } else if (resultadoJogadorB > resultadoJogadorA) {
                vitoriasJogadorB++;
            } else {
                empates++;
            }
        }
        System.out.println(String.format(
                "\nJogador A: %d\nJogador B: %d\nEmpates: %d",
                vitoriasJogadorA, vitoriasJogadorB, empates));
    }
}

public class Principal {

    public static void main(String[] args) {
        Sorteador sorteador = new DadoDeGamao();
        Jogo jogo = new Jogo(sorteador);
        jogo.jogar(100000);
        System.out.println(String.format("Média de pontos por sorteio: %.2f",
                sorteador.getMediaSorteios()));
    }
}
```

```java
public class Dado implements Sorteador {

    protected Random random;
    protected int numeroDeFaces;
    private int contSorteios;
    private long totalSorteadoAcumulado;

    public Dado(int numeroDeFaces) {
        this.random = new Random();
        this.numeroDeFaces = numeroDeFaces;
    }

    protected int produzirInteiroAleatorio() {
        return this.random.nextInt(this.numeroDeFaces) + 1;
    }

    @Override
    public int sortear() {
        int resultado = produzirInteiroAleatorio();
        this.totalSorteadoAcumulado += resultado;
        this.contSorteios++;
        return resultado;
    }

    @Override
    public float getMediaSorteios() {
        return this.totalSorteadoAcumulado / (float) this.contSorteios;
    }

    public int getNumeroDeFaces() {
        return this.numeroDeFaces;
    }
}


/**
 * Esta classe modela o sorteio de números do jogo de gamão.
 * Seu método sortear() deverá produzir um inteiro que é a soma
 * dos resultados obtidos em dois lançamentos de um dado comum.
 * Caso os resultados dos dois lançamentos sejam iguais,
 * deve-se considerar o dobro da soma obtida.
 */
public class DadoDeGamao extends Dado {

    @Override
    protected int produzirInteiroAleatorio() {
        int resultado1 = super.produzirInteiroAleatorio();
        int resultado2 = super.produzirInteiroAleatorio();
        int resultado = resultado1 + resultado2;
        if (resultado1 == resultado2) {
            resultado *= 2;
        }
        return resultado;
    }
}
```

QUESTÃO 1: Se você criasse uma nova classe estendendo a classe Dado, você poderia, dentro dessa classe, acessar o atributo *numeroDeFaces* para leitura*?*
(a) Sim, porque atributos protected são sempre acessíveis a subclasses.
(b) Sim, mas apenas se a subclasse estivesse no mesmo package.
(c) De forma alguma.
(d) Sim, mas apenas via getter method.

QUESTÃO 2: Qual dessas ações acabaria com o erro de compilação na classe DadoDeGamao?
(a) Declarar que ela “implements Sorteador”.
(b) Fazer override em sortear() e getMediaSorteios().
(c) Criar um construtor default em Dado.
(d) Criar um construtor em DadoDeGamao, chamando `super()` em sua primeira linha.

QUESTÃO 3: O que aconteceria se a primeira linha da classe Principal mudasse para
"`Sorteador sorteador = new Sorteador();`” ?
(a) Tomaríamos erro de compilação, porque Sorteador não pode ser instanciado.
(b) Tomaríamos erro de compilação, porque toda classe que implementa Sorteador, no nosso código, exige a passagem de parâmetros para seu construtor.
(c) Tomaríamos erro *em tempo de execução.*
(d) Tudo funcionaria normalmente.

QUESTÃO 4: Escreva um overload para o construtor de Dado em que nenhum parâmetro é passado. O número de faces deverá, nesse caso, ser 6 (isto é, um dado comum).

Daqui para a frente, assuma que o construtor da questão anterior já foi implementado.

QUESTÃO 5: Imagine que incluímos a seguinte linha ao final do construtor da classe Jogo:

```
System.out.printf(“Número de faces = %d”,
     ((Dado)sorteador).getNumeroDeFaces());
```

O que acontecerá durante a execução dessa linha, quando executarmos o main() da classe Principal?
(a) Será impressa uma linha “Número de faces = 6".
(b) Será impressa uma linha “Número de faces = 12".
(c) O código não executará porque teremos um erro de compilação de cast inválido.
(d) Tomaremos um erro de cast inválido em tempo de execução.

QUESTÃO 6: Qual das afirmações abaixo é FALSA?
(a) O atributo “contSorteios" em Dado é privado porque queremos impedir que outros trechos do código modifiquem indevidamente seu valor, já que queremos ter certeza de que ele é incrementado apenas quando um sorteio é feito.
(b) A relação entre a classe Jogo e a interface Sorteador é de *agregação*, já que o Sorteador referenciado por Jogo tem vida própria e sobreviverá ao próprio Jogo.
(c) Nenhum utilizador de um objeto Dado poderá consultar seu número de faces, porque o atributo “numeroDeFaces" é privado e getNumeroDeFaces() não foi declarado na interface Sorteador.
(d) A relação entre Dado e Random é de *composição*, pois cada Dado cria seu próprio gerador de números aleatórios, e a existência deste último está condicionada à existência do primeiro.

QUESTÃO 7: Se modificarmos o tipo do parâmetro esperado no construtor da classe Jogo para ser um Dado, ao invés de um Sorteador, e executarmos o main() da classe Principal, o que acontecerá?
(a) O jogo instanciado no main() usará o DadoDeGamao passado como parâmetro como se este fosse um mero Dado (de 6 faces), e portanto todos os sorteios feitos durante o jogo retornarão algo entre 1 e 6.

(b) A variável “resultadoJogadorA”, no método jogar() de Jogo, receberá o valor 2 com probabilidade zero.
(c) Teremos um erro de compilação, porque no construtor estaremos atribuindo um Dado (passado como parâmetro) ao atributo “sorteador”, que é um Sorteador.
(d) Teremos um erro em tempo de execução.

QUESTÃO 8: Qual seria o efeito de acrescentarmos o modificador “static" no atributo “contSorteios" da classe Dado?
(a) Teríamos erro de compilação, porque esse atributo está sendo usado dentro do método sortear(), que não é static.
(b) Todos os sorteios realizados por todas as instâncias de Dado do nosso sistema incrementariam esse atributo estático.
(c) Teríamos que marcar o método sortear() como static, e aí tudo passaria a funcionar como antes *desde que tivéssemos apenas uma instância de Dado no nosso sistema*.
(d) A cada novo Dado que fosse instanciado, o contSorteios visto por todas as instâncias pré-existentes seria zerado.

QUESTÃO 9: Complete o unit test abaixo:

```
public class DadoTest {

    private Dado dado;

    @Before
    public void setUp() {
        dado = new Dado();
    }

    @Test
    public void testGetMediaSorteios() throws Exception {
        int total = 0;
        for (int i = 1; i <= 1000; i++) {
            total += dado.sortear();
        }
        float mediaEsperada = _________________ / (float) 1000;

        assert__________("A média retornada deve refletir todos os sorteios feitos",

                ____________________________________________________________);
    }
}
```

QUESTÃO 10: Assumindo a existência de todo o código que foi fornecido, escreva abaixo o código para uma classe Moeda, que deve implementar a interface Sorteador, e cujo método sortear() deve retornar 1 ou 2 (fazendo o papel de cara ou coroa). Detalhe: você deve fazer isso *da maneira mais elegante e econômica possível*.