Verificação e Validação de Software I

Teste Baseado em Especificação

Exercícios Implementação JUnit de Casos de Teste por Particionamento

1) Exercício de codificação:

Uma barca de passageiros tem 1200 lugares organizados em 60 fileiras de 20 lugares cada. O sistema de controle de lugares deve controlar tanto a ocupação dos lugares como a distribuição de peso na barca. Desta forma, quando o cliente chega para embarcar ele escolhe um lugar, e o sistema deve dizer se o lugar está ocupado ou se ele não pode se sentar ali em função da distribuição de peso. As regras de distribuição de peso são as seguintes:

- Os primeiros 100 passageiros só podem se sentar nas fileiras de 1 a 20.
- Os próximos 100 passageiros só podem se sentar nas fileiras de 40 a 60.
- Os demais passageiros podem sentar-se em qualquer lugar livre.

Os lugares são identificados da seguinte forma: F<nro da fileira>A<nro do assento>. A numeração das fileiras e lugares inicia em 1. Tanto o assento como a fileira têm de ser informados com 2 dígitos.

Exemplos: F02A12, F45A01, F33A18

A classe *Barca* tem um método chamado *int ocupaLugar(String assento)* que pode retornar um dos seguintes valores:

- 0 – Identificador de assento inválido
- 1 – Assento ocupado
- 2 – Assento bloqueado devido a distribuição de peso
- 3 – Assento atribuído ao passageiro com sucesso

Esqueleto da classe *Barca*:

```
public class Barca {

    public Barca(){
    }

    // Método auxiliar projetado para facilitar testes
    // Ocupa o lugar sem verificação
    protected void ocupaLugarSemVerificacao(int fila, int
assento){
    }

    /*
     * Retorna:
     * 0 – Identificador de assento inválido
     * 1 – Assento ocupado
     * 2 – Assento bloqueado devido a distribuição de peso
     * 3 – Assento atribuído ao passageiro com sucesso
     */
    public int ocupaLugar(String assentoInformado){
    }
```

```
}
```

OBS: o método “ocupaLugarSemVerificação” foi projetado visando testabilidade. Para podermos testar o método “ocupaLugar” teremos de montar cenários diversos, ocupando os lugares de maneira a criar situações para testar se o método “ocupaLugar” está funcionando corretamente. Para facilitar a montagem dos cenários, o método “ocupaLugarSemVerificação” permite que se marque como ocupado um lugar da barca apenas informando a fileira e o número do assento. O método irá marcar o lugar como ocupado sem nenhuma verificação das regras definidas.

Tarefas:

a) Gerar um conjunto de casos de teste para o método “ocupaLugar” utilizando a técnica de particionamento.
b) Usando o JUnit, implementar um driver de teste que exercite o método “ocupaLugar” com os casos de teste definidos na letra “a”.
c) Utilizar a implementação da classe *Barca* (fornecida no Moodle), aplicar o driver de teste sobre a mesma e relatar os defeitos encontrados (se houverem).

2) Exercício de codificação:

Um determinado site web deve armazenar o ranking dos 10 melhores jogadores de um determinado game (ordenados por ordem decrescente de pontuação). Cada vez que uma partida oficial ocorre, o administrador do site entra com o nome do jogador e sua pontuação. Se houverem menos de 10 jogadores cadastrados, o jogador será inserido no ranking independente de seu score. Na medida em que já houver 10 jogadores cadastrados, novos jogadores só entram se tiverem score maior que o último colocado e, sempre que um novo jogador entrar no ranking, o jogador com pontuação mais baixa é eliminado. Os registros devem ser mantidos ordenados em ordem decrescente de pontuação.

Para atender a especificação, um desenvolvedor criou as classes *Record* e *Ranking* cujo esqueleto pode ser visto na sequência:

```
public class Record {
    public Record(String name, int score) {}
    public String getName() {}
    public int getScore() {}
    public String toString() {}
}
```

```
public class Ranking {
     public Ranking() {}

     // Insere novo registro na lista. Mantem ordenação.
     // Retorna true se a inserção foi possível.
     public boolean add(Record record) {}

     // Retorna a quantidade de registros armazenados.
     public int numRecords() {}

     // Retorna o i-ésimo registro armazenado ou
```

```
      // null se o valor de i for inválido.
      public Record getScore(int i) {}

      // Retorna o pior score armazenado.
      // Retorna null se a lista estiver vazia.
      public Record worstScore() {}

      // Retorna o melhor score armazenado.
      // Retorna null se a lista estiver vazia.
      public Record bestScore() {}
}
```

Tarefas:

a) Gerar um conjunto de casos de teste para a classe *Ranking* utilizando a técnica de particionamento (gere um conjunto para cada método).
b) Usando o JUnit, implementar um driver de teste que exercite a classe *Ranking* com os casos de teste definidos na letra "a".
c) Utilizar a implementação da classe *Ranking* (fornecida no Moodle), aplicar o driver de teste sobre a mesma e relatar os defeitos encontrados (se houverem).