Técnicas de Programação, turma 128

Prof. Bernardo Copstein

Lista de Exercícios 02

# Contratos Básicos

1. Especifique a interface Java *IFracao* que modela uma fração matemática . Devem ser previstos métodos tanto para definir como consultar o numerador e o denominador, obter o valor decimal equivalente a fração, bem como uma representação *String* da mesma. Considerando o paradigma de programação por contratos, apresente a interface anotada com JML e uma classe que implementa esta interface.
2. É normal que antes de um vôo o chefe de cabine faça a contagem dos passageiros de maneira a verificar se todos os passageiros listados embarcaram. Para auxiliar nessa contagem normalmente são usados pequenos dispositivos mecânicos com apenas dois botões: um que zera o contador e outro que incrementa o contador. Considerando o paradigma de programação por contratos, apresente uma interface anotada com JML que modele este tipo de dispositivo e uma classe que implementa esta interface.

1. No banco “No$$aGrana” todas as contas correntes são abertas com o status “Comum”. Contas da categoria “Comum” só podem fazer operações simples de depósito e retirada. Já contas da categoria “Especial” desfrutam da seguinte vantagem: toda vez que for efetuado um depósito o valor depositado é acrescido de um “bônus” de R$ 10,00 nos depósitos acima de R$ 1000,00 (ex: se forem depositados R$ 2000,00 serão creditados R$ 2010,00). A conta desfruta do “status” de “Especial” enquanto seu saldo for superior a R$ 10.000,00.
2. Analise o código que segue verifique se ocorreram violações do paradigma de programação por contratos no que se refere a programação defensiva.

|  |
| --- |
| **public** **interface** IPilha {  /\*@ **requires** !isFull();  @ **ensures** size() == **\old**(size())+1;  @ **ensures** top() == v;  @\*/  **void** push(**int** v);  //@ **requires** !isEmpty();  //@ **ensures** size() == **\old**(size());  /\*@ **pure** @\*/ **int** top();    /\*@ **requires** !isEmpty();  @ **ensures** **\result** == **\old**(top());  @ **ensures** size() == **\old**(size())-1;  @\*/  **int** pop();  //@ **ensures** **\result** >= 0;  /\*@ **pure** @\*/**int** size();  /\*@ **pure** @\*/**boolean** isEmpty();  /\*@ **pure** @\*/**boolean** isFull();  } |
| **public** **class** PilhaArray **implements** IPilha {  **public** **static** **final** **int** ***TMAX*** = 20;  **private** **int**[] base;  **private** **int** topo;  **public** PilhaArray() {  base = **new** **int**[***TMAX***];  topo = 0;  }  @Override  **public** **void** push(**int** v) {  **if** (topo == ***TMAX***){  **return**;  }  base[topo] = v;  topo++;  }  @Override  **public** **int** top() {  **return** (base[topo - 1]);  }  @Override  **public** **int** pop() {  **if** (topo == 0){  **return** -1;  }  **int** aux = top();  topo--;  **return** (aux);  }  @Override  **public** **int** size() {  **return** topo;  }  @Override  **public** **boolean** isEmpty() {  **return** size() == 0;  }  @Override  **public** **boolean** isFull() {  **return** (size() == ***TMAX***);  }  } |

1. Considere a interface de um tipo abstrato de dados (TAD) do tipo “fila” (FIFO) definido abaixo e a descrição das funcionalidades previstas para cada um dos métodos. Utilizando seu conhecimento acerca desse tipo de estrutura, complete a especificação do TAD utilizando pré e pós condições especificadas em JML. Acrescente métodos na especificação em caso de necessidade.

|  |
| --- |
| **public** **interface** IFila {  **public** **void** enqueue(**int** v);  **public** **int** dequeue();  **public** **int** first();  **public** **int** last();  **public** **int** size();  **boolean** isEmpty();  **boolean** isFull();  } |

1. Imagine um TAD “fila especial”, que tem a mesma interface e o mesmo comportamento do TAD “fila” descrito na questão anterior, exceto por algumas particularidades: se a “fila” está vazia qualquer número pode ser inserido na mesma. Se o sinal do número a ser inserido for diferente do sinal do número que está no final da “fila” o novo valor só pode ser inserido se a soma dos dois for igual a zero (ou seja, se eles forem iguais em módulo). Caso contrário a “fila” só pode receber números com o mesmo sinal do último inserido. Deve-se cuidar, também, para não inserir valores além da capacidade da fila ou tentar retirar valores da fila vazia. Especifique uma classe capaz de modelar o problema proposto definindo sua interface pública e anotando corretamente as pré e pós condições de cada método usando JML.
2. No sistema de uma universidade, a classe “Aluno” deve armazenar o número de matrícula (inteiro entre 100000 e 999999), o nome do aluno (String, não vazio) e as 3 notas que compõem sua avaliação (as notas são valores “int” pertencentes ao intervalo [0, 100]). O número de matrícula e o nome do aluno são informados pelo construtor. Devem existir métodos para informar as notas (uma por vez), consultar o valor das notas (uma por vez), calcular a média final (média aritmética simples) e um método que retorna “true” se o aluno está aprovado (média final>=60) e “false” caso contrário. Uma nota só pode ser consultada se já tiver sido informada e o método de cálculo da média final só pode ser ativado se as 3 notas já tiverem sido informadas, assim como o método que informa a aprovação ou não do aluno. Defina esta classe. Especifique sua interface e anote corretamente as pré e pós condições de cada método usando JML.