UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

DCC004 - Algoritmos e Estruturas de Dados II

Programação defensiva

Renato Martins

Email: renato.martins@dcc.ufmg.br

https://www.dcc.ufmg.br/~renato.martins/courses/DCC004

Material adaptado de PDS2 - Douglas Macharet e Flávio Figueiredo

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

Introdução

- Direção defensiva

> Direção Defensiva é o ato de conduzir de modo a evitar acidentes, apesar das ações incorretas (erradas) dos outros e das condições adversas (contrárias), que encontramos nas vias de trânsito.
>
> – DETRAN

- Desenvolvimento de software
  - Queremos evitar acidentes (erros)
  - Apesar de ações incorretas (usuários)
  - Em condições adversas (resto do programa)

Introdução

- Programação defensiva
  - Não é ser defensivo sobre sua programação
  - "Eu garanto que funciona!"
- "Garbage in, garbage out”
  - Entradas ruins produzem saídas ruins
  - Colocar a culpa (obrigação) no usuário?
- Bons programadores
  - Se defendem, estilo motoristas

Programação Defensiva

- Forma de design protetivo destinado a garantir o funcionamento contínuo de um software sob circunstâncias não previstas
  - "Garbage in, nothing out"
  - "Garbage in, error message out"
  - "No garbage allowed in"

Programação Defensiva
Robustez vs Corretude

- Robustez
  - Sempre tentar fazer algo que permita que o software continue operando, mesmo que isso às vezes leve a resultados imprecisos
- Corretude (exatidão)
  - Nunca retornar um resultado impreciso
  - Não retornar nenhum resultado será melhor do que retornar um resultado incorreto
- Qual priorizar?

Programação Defensiva
Robustez vs Corretude

- Validação das entradas
- Asserções
- Programação por contrato
- Barricadas
- Tratamento de exceções

Validação das Entradas
Deve ser feita para todo método

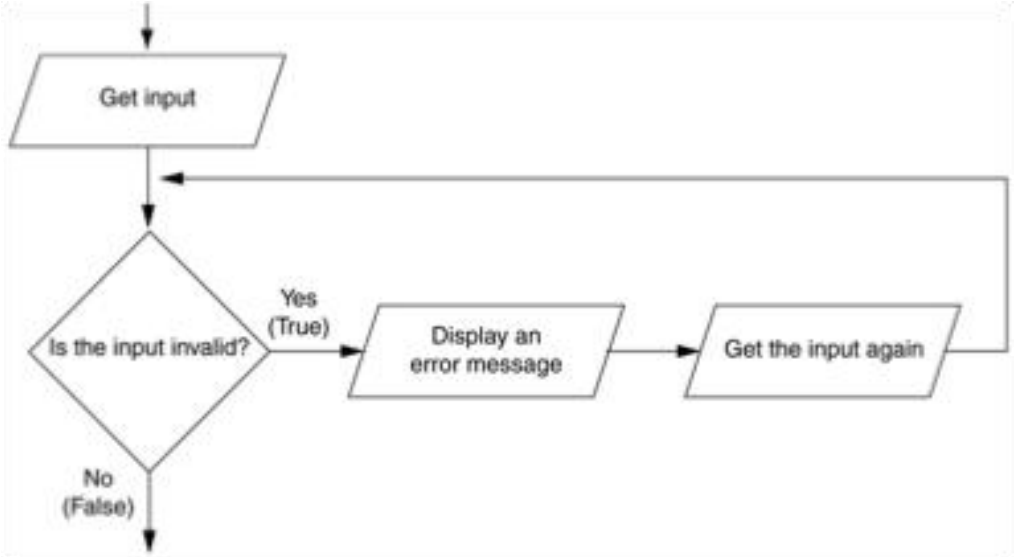
- Quais valores fazem sentido o método receber? Quais tipos?

```
Conta c;
c.depositar('a');
c.depositar(-10);
c.depositar(9.99999);
c.depositar(1.0e-10);
```

- Como devemos agir quando o tipo não é suficiente?
  - No exemplo acima, frações de centavos

Validação das Entradas
No caso do usuário

- Validation loop



- Cada aplicação define tal comportamento
  - Podemos ficar eternamente no loop
  - Sugerir respostas

Asserções

- Asserção/Assertiva
  - Predicado inserido para verificar (certificar) determinada condição (suposição)
- Argumentos
  - Expressão booleana (que deve verdadeira)
  - Mensagem a ser exibida caso não seja
- Podem ser utilizadas no teste ou código

Asserções
Exemplo em Código

```
//#define NDEBUG
#include <iostream>
#include <cassert>

int main() {
  int vetor[10];
  for (int i = 0; i < 15; i++) {
    assert(0 <= i && i < 10);
    vetor[i] = i;
    std::cout << vetor [i] << std::endl;
  }
}
```

Asserções
Código vs Testes

- Nos testes usamos asserções para testar nossas classes como clientes dela
- Dentro do código usamos asserções para indicar erros em tempo de execução
  - Não pegamos asserções
  - Não são tipos Exception
  - Geralmente desabilitada ao lançar o código
- Fail-fast programming

Asserções

Uma boa prática é usar o assert em métodos private

```cpp
#include <cassert>
#include <map>
#include <string>
#include <vector>

class ProcessadorTexto {
private:
  // guarda termo -> número de vezes em um documento
  std::map<std::string, std::vector<int>> _contagens;

  int _soma_um(std::string str) {
    assert(_contagens.count(str));
    int soma = 0;
    for (int count : _contagens[str]) {
      soma += count;
    }
    return soma;
  }
public:
  int soma_total() {
    int soma = 0;
    for (auto pair : _contagens) {
      soma += _soma_um(pair.first);
    }
    return soma;
  }
};
```

Exceções vs Asserções
Quando e como utilizar

- Métodos públicos
  - Validam os dados externos
- Métodos privados
  - Assumem que é seguro usar os dados
- Asserções vs. Exceções
  - Considere o uso de exceções para métodos públicos e asserções para métodos privados

Programação por Contrato

- Acordo entre duas ou mais partes
  - Funções devem ser vistas como um contrato
  - Executam uma tarefa específica
- Não devem fazer outra coisa além disso
- Métodos públicos definem como nossas classes serão utilizadas

Programação por Contrato

- Perguntas
  - O que o contrato espera?
  - O que o contrato garante?
  - O que o contrato mantém?
- Elementos (formalização lógica)
  - {Pré-condições} ação {Pós-condições}
    {Invariantes}

Exemplo

Quão bom é o código abaixo?

```
class Conta {
private:
  int _agencia;
  int _numero;
  double _saldo;
public:
  void sacar(double valor) {
    this->_saldo -= valor;
  }
};
```

Conta Corrente

- Pré-condição
  - Saldo em conta
- Pós condição
  - Redução do saldo **apenas** se houver saldo suficiente para realizar o saque
  - Exceção caso contrário
- Invariante
  - Conta corrente sempre com saldo positivo

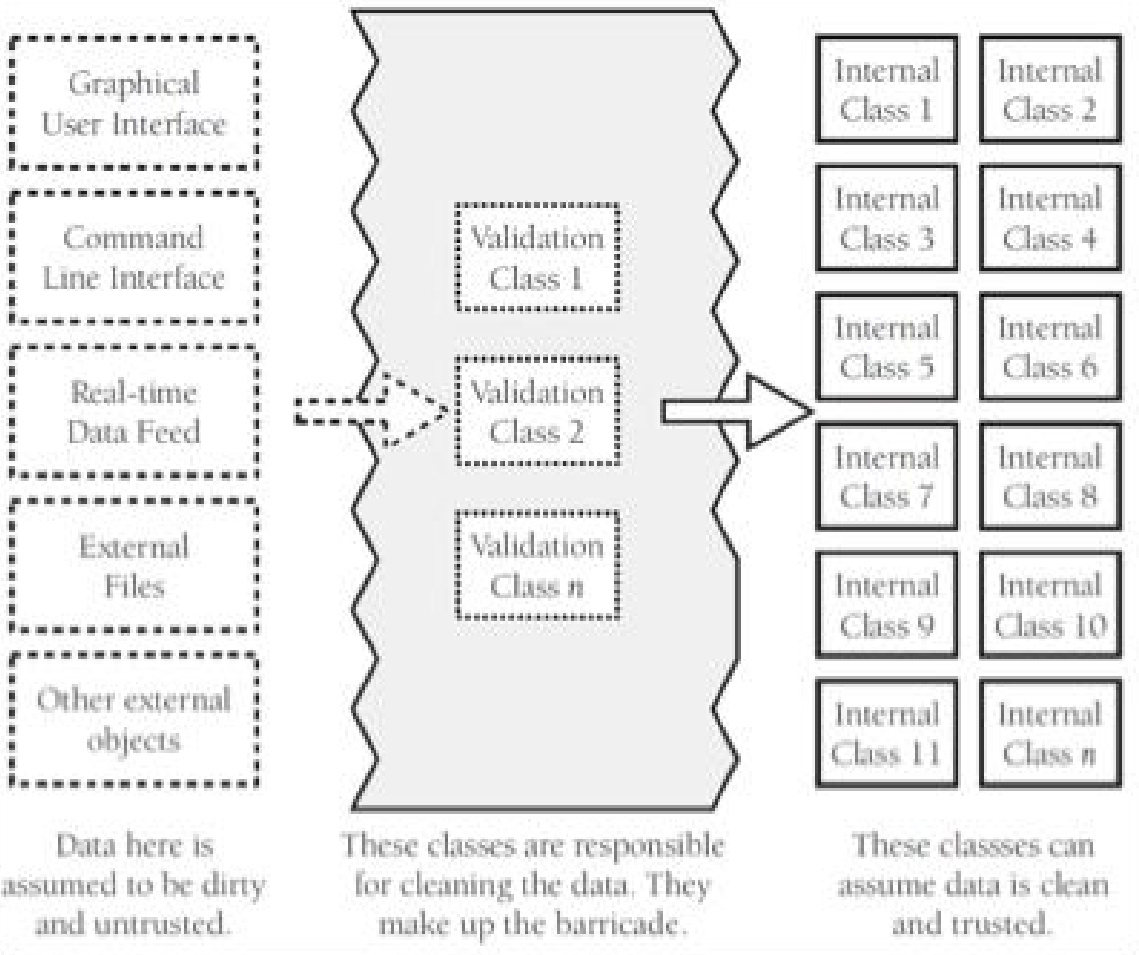
Barricadas

Garantem que o código está correto depois de um certo ponto

- Crie barricadas no programa para minimizar o dano causado por dados incorretos
  - Interfaces como limites para áreas "seguras" Verifique todos os dados que cruzam os limites de áreas seguras
- Partes que funcionam com dados *sujos* e algumas que funcionam com dados *limpos*

Barricadas

Caso de Uso: Programa com diversas interfaces



TDD e Programação Defensiva Caminham juntos

- Test driven development
  - Metodologia de desenvolvimento
  - Teste, código, refactor
- Programação defensiva
  - Boas práticas para ter um código seguro
- TDD **deve** verificar as condições da programação defensiva