Universidade Federal do Rio Grande do Sul — Instituto de Informática
INF01120 – Técnicas de Construção de Programas — 2019/1
Rafael Baldasso Audibert — 00287695

Lista de Exercícios 1

Tipos Abstratos de Dados - TAD

Especifique um TAD com os tipos, operações que representem um

1. *Janela (Parte de um GUI)*
2. *Elevador que seja capaz de transportar diferentes tipos de elementos (parametrizado)*

Além disso, detalhe axiomas e pré-condições

Legenda de Cores:

- Instância da Classe - Verde
- Booleanos - Rosa
- Inteiros - Marrom
- Variável da Instância - Violeta

1. Janela
   - Operações
     - **new:** WINDOW (construtor)
     - **close:** WINDOW $\nrightarrow$ WINDOW
     - **open:** WINDOW $\nrightarrow$ WINDOW
     - **minimize:** WINDOW $\nrightarrow$ WINDOW
     - **maximize:** WINDOW $\nrightarrow$ WINDOW
     - **opened:** WINDOW $\rightarrow$ Boolean
   - Axiomas
     - **maximize(minimize(x))** = maximize(x)
     - **minimize(maximize(x))** = minimize(x)
     - **opened(new)** = True
     - **opened(open(x))** = True

- **opened(close(x))** = False

- Pré-Condições
  - **open**(x: WINDOW) requires NOT opened(x)
  - **close**(x: WINDOW) requires opened(x)
  - **minimize**(x: WINDOW) requires opened(x)
  - **maximize**(x: WINDOW) requires opened(x)

2. Elevador[G]

- Operações
  - **new:** limite: Integer X atual: Integer $\nrightarrow$ ELEVADOR[G] (construtor)
  - **up:** ELEVADOR[G] X Integer $\nrightarrow$ ELEVADOR[G]
  - **down:** ELEVADOR[G] X Integer $\nrightarrow$ ELEVADOR[G]
  - **valid:** ELEVADOR[G] X Integer $\rightarrow$ Boolean
- Axiomas
  - **up(down**(x: ELEVADOR[G], y: INTEGER) = x
  - **valid**(x, y) retorna True se atual + y: Integer $\leq$ limite, cc. False
- Pré-Condições
  - **up**(x: ELEVADOR[G], y: INTEGER) requires valid(x, y)
  - **down**(x: ELEVADOR[G], y: INTEGER) requires valid(x, y)
  - **new**(limite, atual) requires atual $\geq 0 \wedge$ limite $\geq 0 \wedge$ atual $\leq$ limite

Orientação a Objetos - OO

1. *Quais os três recursos característicos de LPs que suportam a Orientação a Objetos?*
   ◇ Objeto, Classe e Mensagem

2. *Qual a diferença entre classe e objeto?*
   ◇ Classe é uma definição abstrata de um objeto (real ou abstrato) do mundo real, que define a estrutura e o comportamento dos mesmos, servindo como um template para a criação da representação armazenada em memória deles
   ◇ Objetos são a instância de uma classe, a representação em memória de um objeto do mundo real

3. *Explique os diferentes níveis de proteção de atributos e métodos em Java e C++*
   ◇ Em C++, a classe é a única unidade de "encapsulamento", por isso necessitamos de mecanismos extras, tais como *funções friends*
   ◇ Em Java, além da classe também temos o conceito de *package* quando falamos de encapsulamento
   ◇ Os tipos de proteção disponíveis são:

   Public Acessível em qualquer lugar do código, inclusive em construtores e destrutores

   Protected Acessível somente por métodos daquela classe, além de classes derivadas

   Private Acessível somente por métodos da própria classe

4. *Explique as diferentes formas de relacionamentos entre classes*
   ◇ Possuímos as seguintes formas de relacionamento:

   Associação Relacionamento estrutural que conecta objetos entre si

   - **Agregação** é o relacionamento *contém-um*, onde os tempos de vida podem ser diferentes, ex.: *Janela e Botão (de uma GUI)*
   - **Composição** é o relacionamento *possui-um*, onde as partes não tem significado sem o todo e vice-versa, onde os tempos de vida são os mesmos, ex.: *Controlador de Temperatura e Sensor*

   Dependência Relacionamento entre duas classes onde uma (classe cliente) usa outra (classe fornecedora), ex.: *classe Line que, para se mover, move um ponto seu, usando uma função definida em uma classe Ponto*

   Generalização Relacionamento que ocorre ao identificarmos classes que possuem propriedades e comportamentos similares, fazendo com que seja interessantes criar uma classe que possua essas propriedas em comum e usar o conceito de Herança para especializarmos as nossas classes, ex.: *classes Carro e Moto que herdam de uma classe VeiculoMotorizado*

Realização Em Java, demonstrando com o uso de *interfaces*, que é uma coleção de operações que especifica um serviço de uma classe ou componente, formalizando o polimorfismo, ex.: *Caso não tenhamos métodos concretos em uma classe Forma que irá generalizar Quadrado e Círculo, é mais interessante apenas definirmos O QUE as classes Quadrado e Círculo deveriam implementar (por exemplo, getArea()) calculando cada uma de sua maneira*

5. *Defina os mecanismos de herança e herança múltipla*
   ◇ Podemos definí-las da seguinte maneira:

   Herança É uma forma de organizar classes com propriedades em comum, agrupando-as de forma que suas propriedades em comum possam ser definidas apenas uma única vez. É um relacionamento *"é um tipo de "*. Esse mecanismo possibilita que uma classe B *herde propriedas* (métodos e variáveis) de uma classe A. Em Java pode ser utilizado definindo uma classe A, e utilizando a seguinte sintaxe na criação da segunda:

```
1  class B extends A {
2      ...
3  }
```

   Herança Múltipla É um conceito de que uma classe poderia herdar de mais do que uma única outra classe. Importante ressaltar que ela não estabelece herança entre múltiplos níveis e nem estabelece relação entre as sub-classes. Existem alguns problemas na real implementação da herança múltipla nas linguagens de programação que em Java são contornados através do conceito de *interface*.

6. *Quais os problemas relacionados ao suporte a herança múltipla? Como Java e C++ resolvem estes problemas?*
   ◇ O principal problema é *Se possuímos um método com a mesma assinatura definido em duas sub-classes (A e B), qual método o objeto do tipo C (que herda tanto de A quanto de B) irá usar?*. Em Java, a solução para esse problema foi não permitir a herança múltipla e introduzir o conceito de *interface*, onde não há ambuiguidade pois a definição do método é feita na classe herdeira, sendo que a interface possui somente a assinatura. Em C++, o programador define qual método utilizar, pois ele redefine o método com a chamada qualificada, permitindo que ele use a cláusula *using* permitindo *overload*.

7. *Explique as regras de compatibilidade entre classes.*
   ◇ Por definição, uma classe A é compatível com A e com qualquer superclasse direta ou indireta de A, isto é objetos de uma sub-classe podem ser usados onde objetos da superclasse são esperados, isso é, podemos fazer uma conversão implícita caso estejamos fazendo um *"upcasting"* na hierarquia. Podemos dizer que Aluno é um subtipo de Pessoa se todas as instâncias de Aluno são, por definição, instâncias de Pessoa, já que tudo que se refere a Pessoa (associações, atributos, operações) também é verdadeiro para Aluno. Por exemplo, o seguinte código é código Java válido:

```
    public class Pessoa{}
    public class Aluno extends Pessoa{}

    public class DemonstraCompatibilidade{
        public void static main(String[] args){
            Pessoa p = new Aluno();
        }
    }
```

8. *Que mecanismo de suporte a LP deve ter para suportar esta regra de compatibilidade?*
   ◇ A LP deverá possuir algum recurso de vinculação dinâmica

9. *O que é um método abstrato? O que é uma classe abstrata? Para que servem estes mecanismos?*
   ◇ Os métodos/classes abstratas serve para sermos capazes de implementar funcionalidades oferecidas pela herança múltipla em Java. Uma classe abstrata, define uma interface comum para uma hierarquia, possui definição dos métodos que existirão em suas subclasses, sendo que todas suas subclasses possuem esses métodos. Um método abstrato é somente a assinatura de um método, restando à classe que herdar a classe abstrata ao qual esse método se encontra, implementá-lo. Também vale ressaltar que não é possível instanciar a classe abstrata e/ou chamar o método abstrato que não foi inicializado.

10. *Explique o conceito de interfaces de Java.*
    ◇ Interface é uma coleção de operações que especifica um serviço de uma classe ou componente, formalizando o polimorfismo, representando agrupamentos de objetos que oferecem os mesmos tipos de serviços ou de operações, não podendo ser instanciadas.

11. *Modifique a classe Conta de forma a adicionar os seguintes atributos, exemplificando os mesmos:*

    - *Agencia*
    - *Número da conta*

12. *Modifique a classe Conta (a partir do exercício anterior) de forma a adicionar os seguintes métodos, exemplificando-os:*

    - *depositar()*
    - *efetuarPagamento()*
    - *colocarCreditosCelular()*

    ◇ Código para as questões 11 e 12, com o seu output:

```
public class Conta {
  private double saldo;
  private int agencia, num_conta;

```

```
5   Conta(double saldo, int agencia, int num_conta){
6     this.saldo = saldo;
7     this.agencia = agencia;
8     this.num_conta = num_conta;
9   }
10
11  public void setSaldo(double saldo) {
12    this.saldo = saldo;
13  }
14
15  public double getSaldo() {
16    return saldo;
17  }
18
19  public int getNum_conta() {
20    return num_conta;
21  }
22
23  public void setNum_conta(int num_conta) {
24    this.num_conta = num_conta;
25  }
26
27  public int getAgencia() {
28    return agencia;
29  }
30
31  public void setAgencia(int agencia) {
32    this.agencia = agencia;
33  }
34
35  public void depositar(double valor) {
36    setSaldo(getSaldo() + valor);
37  }
38
39  public void sacar(double valor) {
40    setSaldo(getSaldo() - valor);
41  }
42
43  public void transferir(Conta destino, double valor) {
44    setSaldo(getSaldo() - valor);
45    destino.setSaldo(destino.getSaldo() + valor);
46  }
47
48  public void imprimir_informacao() {
49    System.out.println("[Agencia " + getAgencia() + "] Saldo da conta " +
      getNum_conta() + " = " + getSaldo());
50  }
51
52  public void efetuarPagamento(String string, double valor) {
53    System.out.println("Efetuando pagamento da conta " + string + " no
    valor de " + valor);
```

```
    sacar(valor);
  }

  public void colocarCreditosCelular(String string, float valor) {
    efetuarPagamento("Creditos para Celular " + string, valor);
  }
}

public class TesteConta {
  public static void main(String[] args) {
    Conta c = new Conta(100, 10504, 1);
    Conta c2 = new Conta(100, 10504, 2);

    System.out.println("Informacoes iniciais");
    c.imprimir_informacao();
    c2.imprimir_informacao();

    System.out.println("Depositando 250 na conta 1");
    c.depositar(250);
    c.imprimir_informacao();

    System.out.println("Transferindo 50 da conta 1 para conta 2");
    c.transferir(c2, 50);
    c.imprimir_informacao();
    c2.imprimir_informacao();


    c.efetuarPagamento("Plano de Saude", 200);
    c.imprimir_informacao();

    c.colocarCreditosCelular("(54) 99645-5959", 50);
    c.imprimir_informacao();

    return;
  }
}
```

Output:

```
Informacoes iniciais
[Agencia 10504] Saldo da conta 1 = 100.0
[Agencia 10504] Saldo da conta 2 = 100.0
Depositando 250 na conta 1
[Agencia 10504] Saldo da conta 1 = 350.0
Transferindo 50 da conta 1 para conta 2
[Agencia 10504] Saldo da conta 1 = 300.0
[Agencia 10504] Saldo da conta 2 = 150.0
Efetuando pagamento da conta Plano de Saude no valor de 200.0
[Agencia 10504] Saldo da conta 1 = 100.0
Efetuando pagamento da conta Creditos para Celular (54) 99645-5959 no
    valor de 50.0
[Agencia 10504] Saldo da conta 1 = 50.0
```

13. *Escreva uma classe Ponto que contém:*

    - *propriedades x e y que podem ser definidos em construtor*
    - *métodos getX() e getY() que retornam x e y*
    - *métodos setX(int) e setY(int) que mudam x e y*

14. *Escreva uma classe Circulo, que contenha:*

    - *raio inteiro e origem Ponto*
    - *construtor que define origem e raio*
    - *método que retorna a área*
    - *método que retorna a circunferência*
    - *use java.lang.Math.PI (Math.PI)*

15. *Crie um segundo construtor para Circulo que aceite um raio do tipo int e coordenadas x e y para origem*

16. *Crie uma classe TestaCirculos que:*

    - *crie um vetor de 5 objetos Circulo*
    - *imprima os valores x, y, raio de cada objeto*
    - *copie a referência do primeiro vetor para o segundo*
    - *crie um terceiro vetor*
    - *copie os objetos do primeiro vetor para o terceiro*
    - *altere os valores de raio para os objetos do primeiro vetor*
    - *imprima os três vetores*

    ◇ Código para as questões 13, 14, 15 e 16 com o seu respectivo output:

```
1  public class Circulo {
2    private int raio;
3    private Ponto origem;
4
5    Circulo(int raio, Ponto origem) {
6      this.setRaio(raio);
7      this.setOrigem(origem);
8    }
9
10   Circulo(int raio, int x, int y) {
11     this.setRaio(raio);
12     this.setOrigem(new Ponto(x, y));
13   }
14
15   public int getRaio() {
```

```
16      return raio;
17    }
18
19    public void setRaio(int raio) {
20      this.raio = raio;
21    }
22
23    public Ponto getOrigem() {
24      return origem;
25    }
26
27    public void setOrigem(Ponto origem) {
28      this.origem = origem;
29    }
30
31    public double getArea() {
32      return Math.PI * Math.pow(getRaio(), 2);
33    }
34
35    public double getCircunference() {
36      return 2 * Math.PI * getRaio();
37    }
38
39    public String toString() {
40      return "[Origem] (" + origem.getX() + ", " + origem.getY() + ") \t [
        Raio] " + getRaio();
41    }
42
43  }
44
45  public class TestaCirculos {
46    public static void main(String[] args) {
47      Circulo[] vetor1 = new Circulo[5];
48      Circulo[] vetor2 = new Circulo[5];
49      Circulo[] vetor3 = new Circulo[5];
50
51      // Criando o primeiro vetor
52      for (int i = 0; i < 5; i++) {
53        vetor1[i] = new Circulo((i + 1) * 2, i, i + 1);
54      }
55      System.out.println("Criacao dos circulos no vetor 1 com os seguintes
        valores:");
56      printaCirculos(vetor1);
57
58      // Copiando a referencia do primeiro vetor para o segundo
59      for (int i = 0; i < 5; i++) {
60        vetor2[i] = vetor1[i];
61      }
62
63      // Fazendo deep copy dos circulos do primeiro vetor para o terceiro
64      for (int i = 0; i < 5; i++) {
```

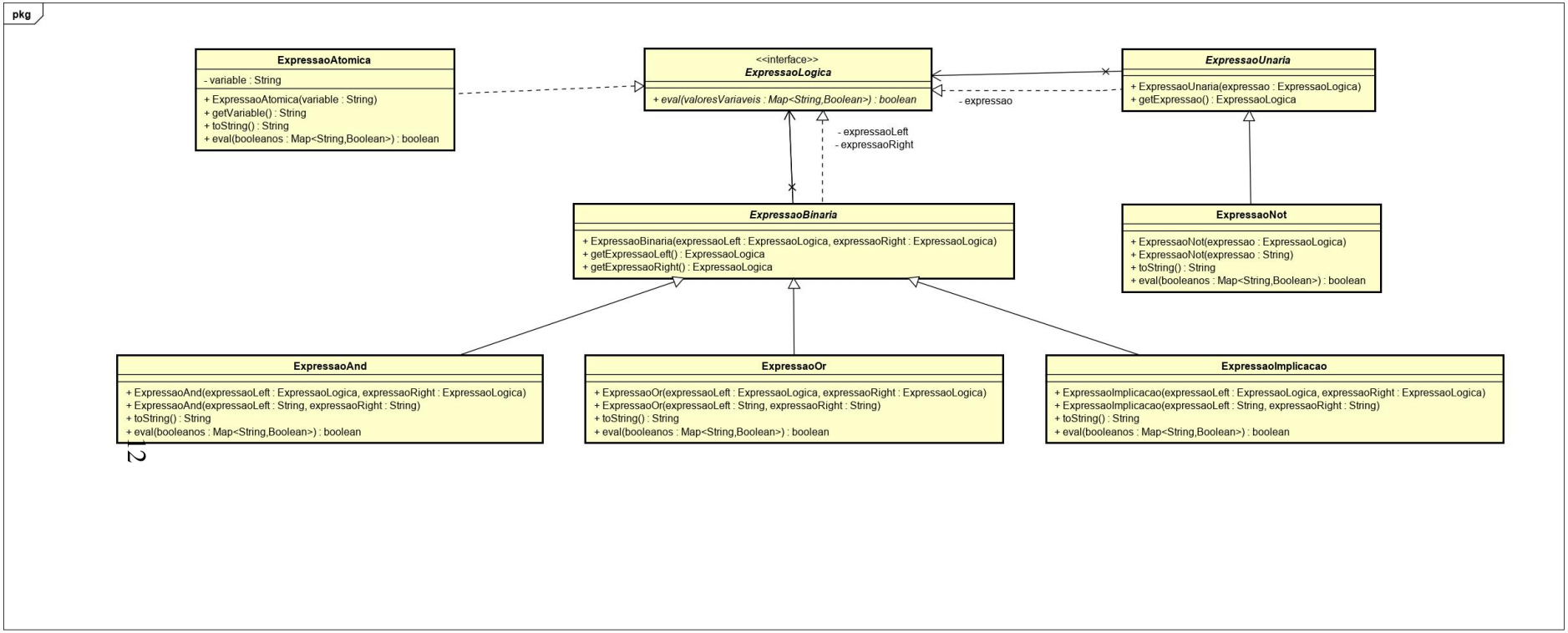
```
65        vetor3[i] = new Circulo(vetor1[i].getRaio(), vetor1[i].getOrigem())
      ;
66      }
67
68      // Alterando valores de raio dos circulos do vetor 1
69      for (Circulo circulo : vetor1) {
70        circulo.setRaio(circulo.getRaio() * 2);
71      }
72
73      // Printando informacoes finais
74      System.out.println("Vetor 1:");
75      printaCirculos(vetor1);
76      System.out.println("Vetor 2:");
77      printaCirculos(vetor2);
78      System.out.println("Vetor 3:");
79      printaCirculos(vetor3);
80    }
81
82    public static void printaCirculos(Circulo[] vetor) {
83      for (Circulo circulo : vetor) {
84        System.out.println(circulo);
85      }
86    }
87  }
```

Output:

```
1  Criacao dos circulos no vetor 1 com os seguintes valores:
2  [Origem] (0, 1)     [Raio] 2
3  [Origem] (1, 2)     [Raio] 4
4  [Origem] (2, 3)     [Raio] 6
5  [Origem] (3, 4)     [Raio] 8
6  [Origem] (4, 5)     [Raio] 10
7  Vetor 1:
8  [Origem] (0, 1)     [Raio] 4
9  [Origem] (1, 2)     [Raio] 8
10 [Origem] (2, 3)     [Raio] 12
11 [Origem] (3, 4)     [Raio] 16
12 [Origem] (4, 5)     [Raio] 20
13 Vetor 2:
14 [Origem] (0, 1)     [Raio] 4
15 [Origem] (1, 2)     [Raio] 8
16 [Origem] (2, 3)     [Raio] 12
17 [Origem] (3, 4)     [Raio] 16
18 [Origem] (4, 5)     [Raio] 20
19 Vetor 3:
20 [Origem] (0, 1)     [Raio] 2
21 [Origem] (1, 2)     [Raio] 4
22 [Origem] (2, 3)     [Raio] 6
23 [Origem] (3, 4)     [Raio] 8
24 [Origem] (4, 5)     [Raio] 10
```

UML

1. *Crie um diagrama de classes que represente o exercício feito na aula prática de introdução à Java, com a implementação de expressões lógicas.*


pkg
ExpressaoAtomica
- variable : String
+ ExpressaoAtomica(variable : String)
+ getVariable() : String
+ toString() : String
+ eval(booleanos : Map<String,Boolean>) : boolean
<<interface>>
ExpressaoLogica
+ eval(valoresVariaveis : Map<String,Boolean>) : boolean
- expressao
ExpressaoUnaria
+ ExpressaoUnaria(expressao : ExpressaoLogica)
+ getExpressao() : ExpressaoLogica
- expressaoLeft
- expressaoRight
ExpressaoBinaria
+ ExpressaoBinaria(expressaoLeft : ExpressaoLogica, expressaoRight : ExpressaoLogica)
+ getExpressaoLeft() : ExpressaoLogica
+ getExpressaoRight() : ExpressaoLogica
ExpressaoNot
+ ExpressaoNot(expressao : ExpressaoLogica)
+ ExpressaoNot(expressao : String)
+ toString() : String
+ eval(booleanos : Map<String,Boolean>) : boolean
ExpressaoAnd
+ ExpressaoAnd(expressaoLeft : ExpressaoLogica, expressaoRight : ExpressaoLogica)
+ ExpressaoAnd(expressaoLeft : String, expressaoRight : String)
+ toString() : String
+ eval(booleanos : Map<String,Boolean>) : boolean
ExpressaoOr
+ ExpressaoOr(expressaoLeft : ExpressaoLogica, expressaoRight : ExpressaoLogica)
+ ExpressaoOr(expressaoLeft : String, expressaoRight : String)
+ toString() : String
+ eval(booleanos : Map<String,Boolean>) : boolean
ExpressaoImplicacao
+ ExpressaoImplicacao(expressaoLeft : ExpressaoLogica, expressaoRight : ExpressaoLogica)
+ ExpressaoImplicacao(expressaoLeft : String, expressaoRight : String)
+ toString() : String
+ eval(booleanos : Map<String,Boolean>) : boolean

Qualidade de Software

1. *Discuta a relação entre os seguintes atributos de qualidade*

   a) *Eficiência e Integridade*

   Eficiência diz respeito ao desempenho do programa, i.e., tempo de resposta do processamento, velocidade de execução das funções. Também diz respeito à quantidade e duração da utilização dos recursos. Enquanto isso, a Integridade se refere a consistencia entre os dados definidos no programa, e a capacidade de tolerancia a falhas do software. Como a Integridade exige cuidados extras por parte do software, ela anda de desencontro com a Eficiência, muitas vezes.

   b) *Eficiência e Adaptabilidade*

   Com Adaptabilidade nos referimos a capacidade do Software se adaptar a outros ambientes por meios e ações próprias, ou seja, é capaz de, sozinho, enfrentar adversidades que possam vir a ocorrer (problemas com sistema operacional, problemas de energia, etc.), levando sempre em consideração que não podemos afetar a eficiência, tanto de desempenho, quanto energética, de um programa.

2. *Como devemos especificar a qualidade de software desejada de tal forma que ela tenha um significado?*

   A qualidade de software deve ser clara, concisa, e mensurável, dessa maneira, ela possui significado, independentemente da maneira da qual ela é definida.

3. *Defina qualidade interna e externa do software, e discuta a sua relação.*

   Qualidade externa de software é aquela qualidade que é vista pelos olhos do usuário, sendo que seus principais fatores são: correção, robustez, extensibilidade, reusabilidade, compatibilidade, eficiência, etc... Quanto a qualidade interna de software, normalmente estamos nos referindo àquilo que é invisível pelo usuário, mas alcançável pelos projetistas e programadores. Se não mantermos uma qualidade interna, que façam com que tenhamos uma fácil modificabilidade, verificabilidade, manutenabilidade, etc, com certeza não seremos capaz de manter a qualidade externa que tanto é buscada.

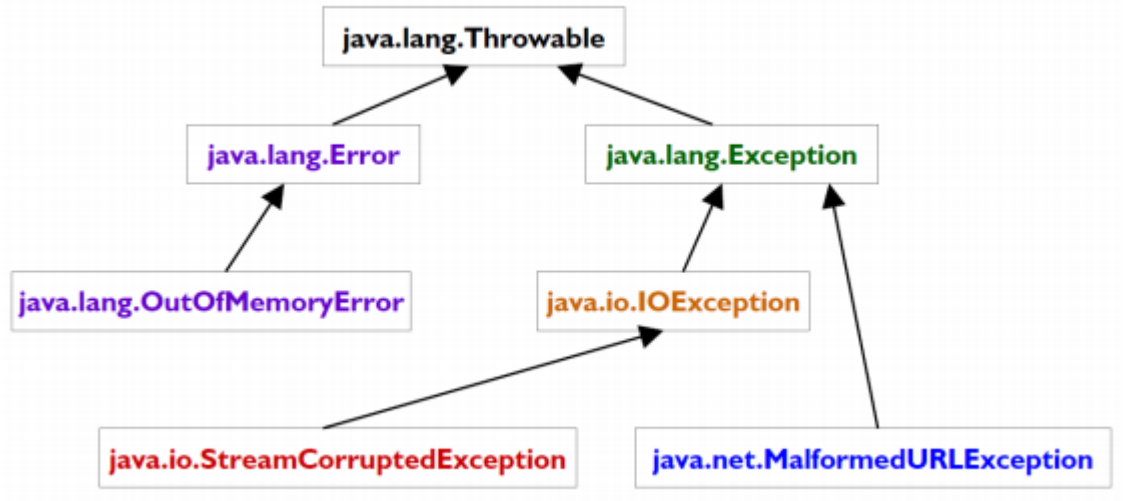
Tratamento de exceções e assertivas

1. *Considere o seguinte código:*

```
try {
    URL u = new URL(s); // s is a previously defined String
    Object o = in.readObject(); //in is a valid ObjectInputStream
    System.out.println("Success");
}
catch (MalformedURLException e) {
    System.out.println("Bad URL");
}
catch (IOException e) {
    System.out.println("Bad file contents")
}
catch (Exception e) {
    System.out.println("General exception");
}
finally {
    System.out.println("Doing finally part");
}
System.out.prinln("Carrying on");

```

Considere a seguinte hierarquia:



(a) *Que linhas são impressas se os métodos das linhas 2 e 3 completarem com sucesso sem provocar exceções?*

A) *Success*
B) *Bad URL*
C) *Bad file contents*
D) *General exception*
E) *Doing finally part*
F) *Carrying on*

(b) *Que linhas são impressas se o método da linha 3 provocar um OutOfMemoryError?*

A) *Success*
B) *Bad URL*
C) *Bad file contents*
D) *General exception*
E) *Doing finally part*
F) *Carrying on*

(c) *Que linhas são impressas se o método da linha 2 provocar uma MalformedURLException?*

A) *Success*
B) *Bad URL*
C) *Bad file contents*
D) *General exception*
E) *Doing finally part*
F) *Carrying on*

(d) *Que linhas são impressas se o método da linha 3 provocar um StreamCorruptedException?*

A) *Success*
B) *Bad URL*
C) *Bad file contents*
D) *General exception*
E) *Doing finally part*
F) *Carrying on*

Assertivas

1. *Implemente um procedimento que tenha um switch para quatro estações (enumeration)*

   - *Em cada case, coloque um System.out.println() que imprima a estação escolhida. Escreva um main que selecione algumas estações.*
   - *Coloque um assert false no default para afirmar que o código jamais deveria passar por lá*
   - *Invente uma quinta estação e veja se o AssertError acontece*

```java
public final class TestaAsserts {

  public enum Seasons { SUMMER, AUTUMN,  WINTER, SPRING, UNKNOWN; }

  public static void main(String args[]) {

    chooseStation(Seasons.SUMMER); // Works just fine, prints Summer
    chooseStation(Seasons.AUTUMN); // Works just fine, prints Autumn
    chooseStation(Seasons.WINTER); // Works just fine, prints Winter
    chooseStation(Seasons.SPRING); // Works just fine, prints Spring

    // AssertionError, APENAS SE RODARMOS O CODIGO COM A FLAG -ea
    chooseStation(Seasons.UNKNOWN);
  }

  public static void chooseStation(Seasons station) {

    switch(station) {
    case SUMMER:
      System.out.println("Summer");
      break;
    case AUTUMN:
      System.out.println("Autumn");
      break;
    case WINTER:
      System.out.println("Winter");
      break;
    case SPRING:
      System.out.println("Spring");
      break;
    default:
      assert(false);
    }
  }
}

```

Projeto de Software visando o reuso

1. *Desenvolver software visando o reuso sempre traz redução de custos? Justifique sua resposta.*

   Na maior parte das vezes sim, já que implementar funções/serviços genéricos facilitar o desenvolvimento de novas funcionalidades, já que podemos nos apoiar em funcionalidades já existentes. Porém, caso estejamos desenvolvendo uma aplicação extremamente específica, que não possuirá novas funcionalidades e/ou não precisará ser refatorada, é um desperdício de dinheiro criar software que visa o reuso, já que é gasto mais tempo para criar software genérico, com boa documentação e testes do que criar um software específico com uma única função.

2. *Faz sentido criar uma classe abstrata que seja final? Justifique sua resposta.*

   Não faz sentido. Classes finais indicam que nós não podemos criar classes filhas, e classes abstratas indicam que devemos criar classes filhas para podermos implementar o método da classe pai, portanto, não faz sentido criar uma classe que cria métodos para suas classes filhas implementarem, porém não podendo ter classes filhas.

3. *Qual a diferença de sobrecarga (overloading) e sobreescrita (overriding).*

   Sobrecarga se refere a quando temos uma função com o mesmo nome, porém com uma assinatura diferente, ou seja, com diferentes tipos de retornos e/ou diferentes parametros (i.e. new Point(x, y) (ponto a partir de x e y) vs. new Point(another_point) (ponto a partir de um outro ponto)).
   Sobreescrita se refere a quando criamos um método em uma classe, com exatamente a mesma assinatura que um método de sua classe pai (i.e. sobreescrita do método toString())

4. *Por que em Java pode-se imprimir diferente tipos de dados (primitivos e objetos) passando os como parâmetro do método cujo nome é System.out.println?*

   System.out.println tem uma implementação específica para cada tipo de dado primitivo. Para os objetos, a classe pai de todos os objetos (Object) tem um método toString() que é chamado pela função System.out.println para obter uma string para printar na tela (sendo que esse método pode ser sobreescrito por qualquer outra classe que herde Object, para podermos customizar o que será printado na tela pelo System.out.println)

5. *Considere o seguinte código:*

```
1 public class HelloWorld {
2     public String toString() {
3         return "HelloWorld!";
4     }
5 }
6
```

```
7  public class CustomizedHelloWorld extends HelloWorld {
8      private String name;
9
10     public CustomizedHelloWorld(String name) {
11         this.name = name;
12     }
13
14     public String toString() {
15         return "HelloWorld, " + name + "!";
16     }
17 }
18
19 public class HelloWorldTest {
20     public static void main(String[] args) {
21         HelloWorld hw = new CustomizedHelloWorld("TCP");
22         new HelloWorldTest().print(hw);
23     }
24
25     public void print(HelloWorld helloWorld) {
26         System.out.println(helloWorld);
27     }
28
29     public void print(CustomizedHelloWorld cHelloWorld) {
30         System.out.println("Ola, mundo!");
31     }
32 }
33
```

(a) *Executando este código, qual o que é impresso na saída padrão?*

A. *Hello World!*

B. *Hello World, TCP!*

C. *Ola, mundo!*

(b) *Justifique por que isso ocorre.*

hw foi criado como um objeto da classe CustomizedHelloWorld (como podemos ver em $new...$), mas armazenado em um objeto da classe HelloWorld, por isso chamamos o método $print$ da classe $HelloWorldTest$ que possua HelloWorld como parâmetro é chamado. Porém, o método System.out.println chama o método toString() da classe mais especializada de hw, no caso $CustomizedHelloWorld$.

6. *Usando Java Generics, crie uma classe Tabela parametrizada pelo tipo T, implementada com um array bidimensional de elementos do tipo T. A classe Tabela deve ter um método que dado uma posição x (linha) e y (coluna) deve retorna um objeto do tipo T. Crie uma classe $TabelaTest$ que mostre o uso da classe $Tabela$.*

```java
import java.util.ArrayList;

public class Tabela<T>{

  private ArrayList<ArrayList<T>> array;

  Tabela(int rows, int columns){

    // Fill the array with null values
    this.array = new ArrayList<ArrayList<T>>(rows);
    for (int i = 0; i < rows; i++) {
      array.add(new ArrayList<T>(columns));

      for (int j = 0; j < columns; j++) {
        array.get(i).add(null);
      }
    }
  }

  T get(int row, int column) {
    return array.get(row).get(column);
  }

  void set(T value, int row, int column) {
    array.get(row).set(column, value);
  }
}

public class Generics {

  static final int ROWS = 5;
  static final int COLUMNS = 5;

  public static void main(String args[]) {
    Tabela<Integer> tabela = new Tabela<Integer>(ROWS, COLUMNS);

    for(int i = 0; i < ROWS; i++) {
      for(int j = 0; j < COLUMNS; j++) {
        tabela.set(i * ROWS + j, i, j);
      }
    }

    for(int i = 0; i < ROWS; i++) {
      for(int j = 0; j < COLUMNS; j++) {
        System.out.println(tabela.get(i, j));
      }
    }
  }
}

```

7. *É possível no código da Tabela fazer "$newT[lin][col]$", onde lin e col são inteiros? Investigue por que e explique a resposta.*

   Não é possível, e isso acontece porque classes genéricas foram implementados em Java de uma maneira que elas NÃO SABEM com que tipo elas foram criadas em runtime, portanto elas não conseguem fornecer type-safety pra array manter os argumentos corretamente dentro dela. Fonte: `https://stackoverflow.com/questions/529085/how-to-create-a-generic-array-in-java`

Convenções e Boas Práticas de Programação

1. *Analise o código disponibilizado na aula 01 e aponte problemas, de acordo com o que foi visto em aula.*

   Os itens abaixos seguem a forma: linhaInicial:linhaFinal — explicação

   - 14 — Nome da struct sem significado, renomear
   - 15 — Nome da variável sem significado, renomear
   - 16 — Nome da variável sem significado, renomear
   - 20 — Nome da variável sem significado, renomear
   - 21 — Nome da variável sem significado, renomear
   - 22 — Nome da variável sem significado, renomear
   - $29 : 33 + 60 : 64$ — Repetição de código, deveria ser abstraído para uma função, ou alterar o $while$ 35:58 por um $do\ while$, possivelmente em uma função própria
   - $30 : 32 + 61 : 63$ — Remover magic numbers, utilizando $enum$
   - $36 : 58$ — Alterar código de $if + else\_if$ para um $switch$, provavelmente mostrando algum erro caso fosse inserido um valor inválido, e mover isso para uma função própria
   - $37 : 40 + 46 : 49$ — Repetição de código, deveria ser abstraído para uma função
   - $43 : 44 + 55 : 56$ — Repetição de código, deveria ser abstraído para uma função