INSTRUÇÕES SOBRE RESPOSTAS:

- **QUESTÃO 1 –** Usar lápis ou caneta e implementar o código manuscrito.
- **QUESTÕES 2, 3 e 4 –** Indicar a linha por referência cruzada. Colocar um sequencial crescente em cada linha com erro no código e, na resposta, colocar o sequencial, em seguida o problema encontrado (não compila, não roda), e finalmente a descrição da causa do problema. Exemplo: na linha Classe1 x = new Classe2() foi encontrado um erro de compilação. Marca-se ***(1)*** Classe1 x = new Classe2() e na resposta (*1) – não compila – bla bla bla justificativa*.
- **QUESTÃO 5 –** Transcrever o que a execução do programa envia para o console de saída NA ORDEM em que os comandos de saída são executados.

QUESTÃO 1 (2,5 PONTOS): Implemente as seguintes classes especificadas abaixo:

a. Duas exceções checadas ExcecaoPrecoInvalido e ExcecaoProdutoInvalido com construtores sem parâmetros.
b. Uma classe ValidadorPreco com um método sem retorno (void) validarPreco, que deve receber como parâmetros um preço e um código de produto (um número inteiro). O método deve:
    i. Lançar ExcecaoPrecoInvalido se o parâmetro preço for menor que zero, ou se o parâmetro preço for maior que 5000 e o código do produto for maior ou igual a 6060, ou se o parâmetro preço for menor que 0,11 e o código do produto estiver entre 1000 e 4999.
    ii. Lançar ExcecaoProdutoInvalido se o parâmetro código do produto for menor ou igual a zero.
c. Uma classe ProgramaValidadorPreco. Deve ser um programa JAVA que leia do teclado um preço e um código do produto, e que, usando o método implementado no item b, imprima no console "Preço/produto OK" se os preço e código do produto lidos forem validados com sucesso e uma das mensagens "Preço inválido" ou "Código do produto inválido" se o preço e/ou código do produto lido for inválido.

OBS: Para ler números do teclado, ver as diretrizes abaixo:

- Definir o seguinte atributo na classe ProgramaValidadorPreco.
    - `private static final Scanner ENTRADA = new Scanner(System.in);`
- Usar o método **`ENTRADA.`**`nextInt()` para ler números inteiros.
- Usar o método **`ENTRADA.`**`nextDouble()` para ler números reais.

REFERÊNCIA NO MATERIAL

Assunto da 1ª unidade; POO – Heranca.pdf; POO – exceções.pdf, pags. 10-19, 36-37.

EXPLICAÇÃO SOBRE A SOLUÇÃO

- O que foi pedido na letra (a) é simplesmente a criação de duas exceções CHECADAS, que, por definição do JAVA, devem herdar de `Exception` e, como foram solicitados construtores vazios, o código dispensa a criação explícita destes.
- O que foi pedido na letra (b) é a implementação de um método que deve sinalizar situações excepcionais por lançamento de exceções, no caso, as duas criadas na letra (a). O lançamento de exceções exige o comando **`throw`** seguido de uma referência a um objeto cujo tipo é subclasse de `Exception`. Normalmente, isto se faz com um **`new`**. Todo método que lança exceções checadas deve, na sua assinatura, declarar as exceções lançadas através da cláusula **`throws`**. A lógica do método leva em conta as situações excepcionais, onde o método sai com lançamento de exceções, e as situações normais onde o método, por ser **`void`**, termina naturalmente seu fluxo de execução.
- O que foi pedido na letra (c) é a criação de um programa que lê do teclado números real e inteiro, e que chamasse o método criado na letra (b). O chamador de um método que declara exceções checadas deve tratar estas exceções com blocos **`try...catch`**, onde são implementados respectivamente os fluxos normais e excepcionais referentes ao retorno do método chamado. Para cada exceção declarada no método chamado deve haver um catch correspondente.

CRITÉRIO DE PONTUAÇÃO

- Letra (a): vale até 0,6, e a resposta é bem simples. Sintaxe errada de definição de classe (opcionalmente de construtor) terá de 0,1 a 0,2 descontados. A não declaração da herança de `Exception` implica na perda de 0,3 por exceção, pois trata-se do cerne da implementação.
- Letra (b): vale até 1,2, e a resposta tem foco na estrutura lógica do método e na sintaxe / forma dos lançamentos e declaração de exceções. Erros básicos de sintaxe na definição do método e na estrutura da classe terão de 0,1 a 0,5 descontados. Erros nos comandos de lançamento de exceções 0,3 a 0,5. Erros na / ausência de declaração das exceções terão 0,2 a 0,5 descontados. Erros na lógica "de negócio" terão 0,3 a 0,6 descontados.
- Letra (c) vale até 0,7 e a resposta tem foco na chamada do método implementado na letra (b). Erros básicos de sintaxe na definição do **`main`** e na estrutura da classe terão de 0,1 a 0,3 descontados. Erros no / ausência de blocos **`try...catch`** das terão 0,3 a 0,5 descontados.

RESPOSTA

```
// opcionalmente pode se ter construtores vazios definidos nas exceções
public class ExcecaoPrecoInvalido extends Exception {}
public class ExcecaoProdutoInvalido extends Exception {}
public class ValidadorPreco {
	public void validarPreco(double preco, int codigoProduto)
		throws ExcecaoPrecoInvalido, ExcecaoProdutoInvalido {
		if ((preco < 0)
			|| (preco > 5000 && codigoProduto >= 6060)
			|| (preco < 0.11 && codigoProduto >= 1000
			&& codigoProduto <= 4999)) {
			throw new ExcecaoPrecoInvalido();
		} // opcionalmente pode se ter um else if
		if (codigoProduto <= 0) {
			throw new ExcecaoProdutoInvalido();
		}
	}
}
public class ProgramaValidadorPreco {
	private static final Scanner ENTRADA = new Scanner(System.in);
	public static void main(String[] args) {
		ValidadorPreco val = new ValidadorPreco();
		double preco = ENTRADA.nextDouble();
		int codigoProduto = ENTRADA.nextInt();
		try {
			val.validarPreco(preco, codigoProduto);
			System.out.println("Preço/produto OK");
		} catch (ExcecaoPrecoInvalido e) {
			System.out.println("Preço inválido");
		} catch (ExcecaoProdutoInvalido e) {
			System.out.println("Código do produto inválido");
		}
	}
}
```

QUESTÃO 2 (1,0 PONTO): Dado o código abaixo, quais linhas possuem erros de compilação ou de execução? Descreva as causas dos erros apontados.

```
public class Veiculo {}
public class Aereo extends Veiculo {}
public class Balao extends Aereo {}
public class Aviao extends Aereo {}
public class Aquatico extends Veiculo {}
public class Lancha extends Aquatico {}
public class ProcessamentoConversao {
	public void rodarConversoes1() {
		Aereo a1 = new Aviao();
		Aquatico aq = new Lancha();
		Veiculo v = new Veiculo();
		Balao b = new Balao();
		aq = v; // ERRO 1
		Lancha la = a1; // ERRO 2
		a1 = b;
	}
	public void rodarConversoes2() {
		Aereo a1 = new Aviao();
		Veiculo v = new Balao();
		Aquatico aq = new Lancha();
		aq = (Lancha)v; // ERRO 3
		Aereo a2 = (Aereo)aq; // ERRO 4
		Balao b1 = (Aereo)v; // ERRO 5
		Aereo a3 = (Balao)v;
	}
}
```

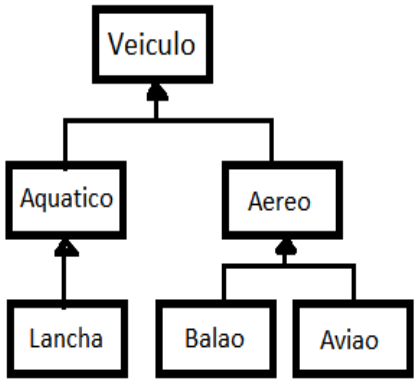
REFERÊNCIA NO MATERIAL

POO – Heranca.pdf; POO – Polimorfismo_PrincipioSubstituicao.pdf, pags. 9-13 e 16-20.

EXPLICAÇÃO SOBRE A SOLUÇÃO

É imprescindível visualizar a hierarquia em questão (dada abaixo) e desenhar o mapa de memória para conversões explícitas, pois é necessário avaliar os tipos dos objetos referenciados. A solução passa por aplicar, a cada linha de código que contém conversão implícita e explícita de tipos não primitivos as regras aplicáveis de complicação e de execução, a saber:

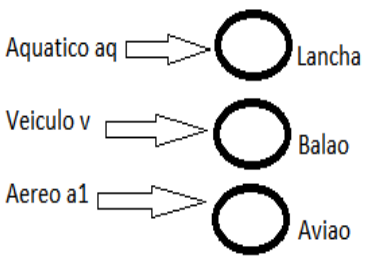
- Em uma conversão implícita, o código compila ou não compila. Compila se o tipo da expressão (variável) de origem for igual ou subtipo do tipo da variável de destino.
- Em uma conversão explícita (CAST), o código pode compilar ou não, e se compilar, pode rodar ou não. O código compila se: o tipo do CAST e o tipo da expressão (variável) de origem estiverem na mesma hierarquia **E** se o tipo do CAST for igual ou subtipo do tipo da variável de origem. O código roda se o tipo do objeto referenciado pela expressão (variável) de origem for igual ou subtipo do tipo do CAST.



A – tipo da variável de destino
B – tipo da variável de origem
C – tipo do cast
D – tipo do objeto referenciado

B b = new D();
...
A a = (C)b

Mapa de memória do método rodarConversoes2(), onde estão as conversões explícitas



RESPOSTA (linhas com erro marcadas em vermelho acima no código)

ERRO 1 – Não compila porque o tipo da variável de origem ***v*** *(**Veiculo**) não é igual ou subtipo do tipo da variável de destino* ***aq*** *(**Aquatico**).*

ERRO 2 – Não compila porque o tipo da variável de origem ***a1*** *(**Aereo**) não é igual ou subtipo do tipo da variável de destino* ***la*** *(**Lancha**).*

ERRO 3 – Não roda porque o tipo do objeto referenciado pela variável de origem ***v*** *(**Balao**) não é igual ou subtipo do tipo do CAST (**Lancha**).*

*ERRO 4 – Não compila porque o tipo do CAST (**Aereo**) e o tipo da variável de origem* ***aq*** *(**Aquatico**) não estão na mesma hierarquia.*

*ERRO 5 – Não compila porque o tipo do CAST (**Aereo**) não é igual ou subtipo do tipo da variável de destino* ***b1*** *(**Balao**).*

CRITÉRIO DE PONTUAÇÃO

Na questão há 05 linhas com erros de compilação. Cada erro identificado e corretamente justificado vale 0.2. Há desconto de 0,2 por linha incorreta apontada. Se não fosse assim, poderia se apontar todas as linhas como incorretas e sempre pontuar potencialmente pelo máximo possível.

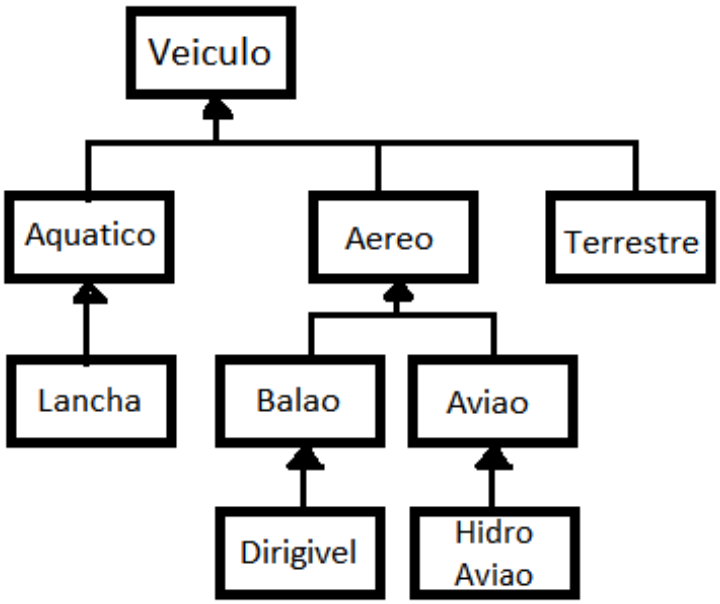
QUESTÃO 3 (1,0 PONTO): Dado o código abaixo, quais linhas possuem erros de compilação? Descreva as causas dos erros apontados.

```
public abstract class Veiculo {
	public abstract void ligar();
	public abstract void desligar();
}
public abstract class Aereo extends Veiculo {
	public void desligar() {
		System.out.println("DESLIGAR AE");
	}
}
public abstract class Aquatico extends Veiculo {}
public final class Aviao extends Aereo {
	public void ligar() {
		System.out.println("LIGAR AV");
	}
	public void desligar() {
		System.out.println("DESLIGAR AV");
	}
}
public class Balao extends Aereo {
	public final void ligar() {
		System.out.println("LIGAR BL");
	}
}
public class Dirigivel extends Balao { // ERRO 1
	public void ligar() {
		System.out.println("LIGAR DR");
	}
}
public class Hidroaviao extends Aviao {} // ERRO 2
public class Lancha extends Aquatico { // ERRO 3
	public void ligar() {
		System.out.println("LIGAR LA");
	}
}
public abstract final class Terrestre extends Veiculo { // ERRO 4
	public void rodar(); // ERRO 5
}
```

REFERÊNCIA NO MATERIAL
POO – Heranca.pdf; POO – Polimorfismo_ClassesAbstratas.pdf, pags. 7-17.

EXPLICAÇÃO SOBRE A SOLUÇÃO
É imprescindível visualizar a hierarquia em questão (dada abaixo). O código deve ser analisado sob a luz das amarrações de compilação envolvendo os conceitos de classes abstratas e do modificador de acesso final.
O código não compila se…

- Uma classe abstrata for instanciada (new…);
- Um método abstrato definido em uma superclasse abstrata não for sobrescrito em uma subclasse concreta (diretamente ou por herança da hierarquia superior);
- Houver chamada, em uma subclasse, via super, de um método definido como abstrato em uma superclasse, mas que não tenha implementação encontrada na super hierarquia;
- Um método abstrato for definido em uma classe concreta;
- Um método abstrato tiver corpo;
- Um método não abstrato não tiver corpo;
- Um método abstrato for final;
- Uma classe abstrata for final;
- Uma classe tentar herdar de outra classe final;
- Uma classe tentar sobrescrever um método final de uma de suas superclasses.



RESPOSTA (linhas com erro marcadas em vermelho acima no código)
*ERRO 1 – Não compila porque o método **ligar()** da classe **Dirigivel** não pode sobrescrever o seu homônimo final da superclasse **Balao**.*
*ERRO 2 – Não compila porque a classe **Hidroaviao** não pode herdar de uma classe final (**Aviao**).*
*ERRO 3 – Não compila porque a classe **Lancha** tem que implementar o método **desligar()**.*
*ERRO 4 – Não compila porque a classe **Terrestre** não pode ser abstrata e final.*
*ERRO 5 – Não compila porque o método **rodar()** é concreto (ou não abstrato) e não tem implementação.*

CRITÉRIO DE PONTUAÇÃO
Na questão há 05 linhas com erros de compilação. Cada erro identificado e corretamente justificado vale 0.2. Há desconto de 0,2 por linha incorreta apontada. Se não fosse assim, poderia se apontar todas as linhas como incorretas e sempre pontuar potencialmente pelo máximo possível.

QUESTÃO 4 (1,0 PONTO): Dado o código abaixo, quais linhas possuem erros de compilação? Descreva as causas dos erros apontados.

```
public class ExcecaoConteudo extends Exception {}
public class ExcecaoModeloGenerico extends Exception {}
public class ExcecaoValidadeInformacao extends ExcecaoModeloGenerico {}
public class ProcessamentoExcecao1 {
	private void verificar1(String str) throws ExcecaoValidadeInformacao {
		if (str == null || str.equals("")) {
			throw new ExcecaoValidadeInformacao();
		}
	}
	private void verificar2(String str) throws ExcecaoConteudo {
		if (str != null && str.length() > 1000) {
			throw new ExcecaoConteudo();
		}
	}
	public void rodar1(String str) throws Exception {
		verificar2(str);
	}
	public void rodar2(String str) {
		try {
			verificar1(str);
		} catch (ExcecaoModeloGenerico e) {
			System.out.println("ExcecaoModeloGenerico");
		} catch(ExcecaoValidadeInformacao e) { // ERRO 1
			System.out.println("ExcecaoValidadeInformacao");
		}
	}
	public void rodar3(String str) throws ExcecaoModeloGenerico, ExcecaoConteudo {
		verificar2(str);
	}
	public void rodar4(String str) {
		try {
			verificar1(str);
		} catch(ExcecaoValidadeInformacao e) {
			System.out.println("ExcecaoValidadeInformacao");
		} catch(ExcecaoConteudo e) { // ERRO 2
			System.out.println("ExcecaoConteudo");
		}
	}
	public void rodar5(String str) { // ERRO 3 – pode ser esta linha
		verificar1(str); // ERRO 3 – ou esta linha
	}
	public void rodar6(String str) {
		try {
			verificar1(str);
			verificar2(str); // ERRO 4
		} catch(ExcecaoValidadeInformacao e) {
			System.out.println("ExcecaoValidadeInformacao");
		}
	}
	public void rodar7(String str) {
		try {
			verificar2(str);
			verificar1(str);
		} catch(ExcecaoModeloGenerico e) {
			System.out.println("ExcecaoModeloGenerico");
		} catch(ExcecaoConteudo e) {
			System.out.println("ExcecaoConteudo");
		}
	}
}
```

```java
public class ProcessamentoExcecao2 {
	public void verificar01(String valor) { // ERRO 5 – pode ser esta linha
		if (valor == null || valor.length() > 1000) {
			throw new ExcecaoValidadeInformacao(); // ERRO 5 – ou esta linha
		}
	}
	public void verificar02(int valor)
			throws ExcecaoModeloGenerico, ExcecaoConteudo {
		if (valor <= 0) {
			throw new ExcecaoValidadeInformacao();
		}
	}
}
```

REFERÊNCIA NO MATERIAL
POO – Heranca.pdf; POO – exceções.pdf, pags. 10-33, 36-37.

EXPLICAÇÃO SOBRE A SOLUÇÃO
É importante observar que **ExcecaoValidadeInformacao** é subclasse de **ExcecaoModeloGenerico**. A questão exige conhecimentos dos mecanismos de LANÇAMENTO (**`throw`**), DECLARAÇÃO (**`throws`**) e TRATAMENTO/REPASSE (**`try...catch e throws`**) de EXCEÇÕES do JAVA, e das amarrações de compilação relativas ao modelo de exceções. O código não compila se…

- Em um método: uma exceção de um tipo LANÇADA (comando **throw**) não tiver seu próprio tipo ou algum supertipo DECLARADA (cláusula **throws**);
- Em um chamador de um método: uma exceção de um tipo DECLARADA no método chamado (cláusula **throws**) não tiver tal tipo ou algum supertipo TRATADO (bloco **catch**) E não tiver tal tipo ou algum supertipo DECLARADO (cláusula **throws** no método chamador);
- Em um chamador de um método: se um método chamado DECLARAR (cláusula **throws**) um subtipo e o método chamador TRATAR (bloco **catch**) primeiro um supertipo e depois um subtipo em blocos **catch** subsequentes;
- Em um chamador de um método: algum tipo for TRATADO (bloco **catch**) mas não tiver, o tal tipo ou algum subtipo, DECLARADO (cláusula **throws** no método chamado).

RESPOSTA (linhas com erro marcadas em vermelho acima no código)
*ERRO 1 – Não compila porque uma super exceção (**ExcecaoModeloGenerico**) é tratada ANTES de uma sub exceção (**ExcecaoValidacaoInformacao**).*
*ERRO 2 – Não compila porque uma exceção (**ExcecaoConteudo**) não declarada no método chamado **verificar1()** é tratada no método chamador **rodar4()**.*
*ERRO 3 – Não compila porque o método chamador **rodar5()** não declara nem trata uma exceção (**ExcecaoValidacaoInformacao**) declarada no método chamado **verificar1()**.*
*ERRO 4 – Não compila porque o método chamador **rodar6()** não declara nem trata uma exceção (**ExcecaoConteudo**) declarada no método chamado **verificar2()**.*
*ERRO 5 – Não compila pois o método **verificar01()** LANÇA uma exceção (**ExcecaoValidadeInformacao**) mas não a DECLARA.*

CRITÉRIO DE PONTUAÇÃO
Na questão há 05 linhas com erros de compilação. Cada erro identificado e corretamente justificado vale 0.2. Há desconto de 0,2 por linha incorreta apontada. Se não fosse assim, poderia se apontar todas as linhas como incorretas e sempre pontuar potencialmente pelo máximo possível.

QUESTÃO 5 (1,0 PONTO): Dado o código abaixo, indique as saídas no console quando executamos o programa em questão.

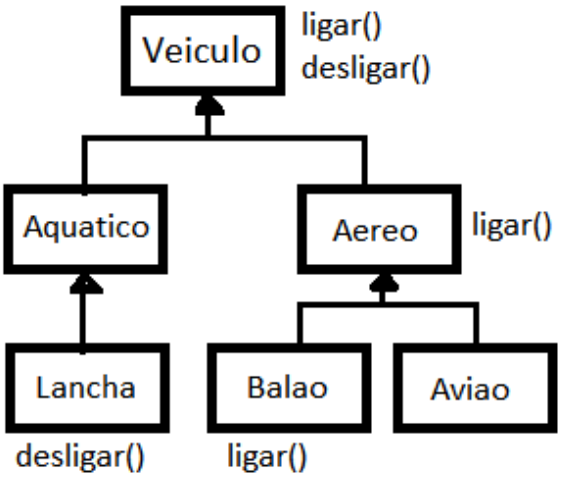
```
public class Veiculo {
	public void ligar() {
		System.out.println("LIGAR VEICULO");
	}
	public void desligar() {
		System.out.println("DESLIGAR VEICULO");
	}
}
public class Aquatico extends Veiculo {}
public class Lancha extends Aquatico {
	public void desligar() {
		System.out.println("DESLIGAR LANCHA");
	}
}
public class Aereo extends Veiculo {
	public void ligar() {
		System.out.println("LIGAR AEREO");
	}
}
public class Aviao extends Aereo {}
public class Balao extends Aereo {
	public void ligar() {
		System.out.println("LIGAR BALAO");
	}
}
public class ProgramaVeiculo {
	public static void main(String[] args) {
		Veiculo v1 = new Aereo();
		v1.ligar();
		v1.desligar();
		Veiculo v2 = new Lancha();
		v2.ligar();
		v2.desligar();
		Balao v3 = new Balao();
		v3.ligar();
		v3.desligar();
		Veiculo v4 = new Aquatico();
		v4.ligar();
		v4.desligar();
		Aereo v5 = new Aviao();
		v5.ligar();
		v5.desligar();
	}
}
```

REFERÊNCIA NO MATERIAL

POO – Heranca.pdf; POO – Polimorfismo_Sobrescrita.pdf, pags. 7-14.

EXPLICAÇÃO SOBRE A SOLUÇÃO

É importante montar a hierarquia em questão e identificar, em cada classe, que métodos estão implementados e/ou reimplementados por sobrescrita. A regra de sobrescrita diz que um método definido e implementado em uma superclasse pode ser definido (com a mesma assinatura) na subclasse e nela reimplementado. Pelo princípio da substituição, é possível referenciar objetos de subtipos com variáveis de supertipos. Através da variável de um supertipo, é possível “enxergar” o método, pois ele foi definido na superclasse. Mas, quando o método é chamado, a “versão” executada é a primeira encontrada na hierarquia, quando esta é percorrida a partir do tipo do objeto referenciado para cima. A solução consiste em aplicar às chamadas dos métodos estas premissas.



RESPOSTA
LIGAR AEREO
DESLIGAR VEICULO
LIGAR VEICULO
DESLIGAR LANCHA
LIGAR BALAO
DESLIGAR VEICULO
LIGAR VEICULO
DESLIGAR VEICULO
LIGAR AEREO
DESLIGAR VEICULO

CRITÉRIO DE PONTUAÇÃO
No código apresentado existem 10 Sysouts. Cada Sysout correto, NA ORDEM CORRESPONDENTE à ORDEM em que aparecem os Sysouts no código, vale 0,1. Se mais de 10 saídas forem indicadas, as saídas de ordem superior a 10 serão ignoradas. Se menos de 10 saídas forem indicadas, serão avaliadas até a quantidade apresentada, com pontuação eventualmente proporcional.