Nome: Vinícius Andrade Frederico RA: 139223

Obs: O arquivo lab05.zip contém o código por completo. Abaixo estão somente os códigos necessários para o relatório.

Item 01)

Ao alterar os tipos das variáveis para int, byte, byte, o programa compila sem problemas. Ao alterar para float, o programa não compila, como já era esperado, uma vez que não há construtor que utilize parâmetros float, float, float.

Item 02)

a) Construtores e funções para obter titular, numConta e senha:

```
public ContaCor(String nome, int num, int pwd){
        titular = nome;
        numConta = num;
        senha = pwd;
        estado = 1; // conta é ativada quando criada
}
public ContaCor(String nome, double val, int num, int pwd) {

        this(nome, num ,pwd);
        saldoAtual = val;

} // fim do construtor
public String getTitular(){
        return titular;
}
public int getNumConta(){
        return numConta;
}
public int getSenha(){
        return senha;
}
```

b) Pelo resultado obtido, pode-se concluir que ao utilizar o construtor sem passar saldo, será gerado uma conta com saldo 0, o mesmo acontece ao utilizar o construtor passando saldo 0. Ou seja, se passarmos um saldo, ele será o saldo utilizado para o objeto, caso contrário, será 0.

ExemploDoisConstrutores.java

```
package br.unicamp.ic.mc302.contaCor;
public class ExemploDoisConstrutores {
        public static void main(String args[]) {
                String titular = "Bob";
                int numConta = 30;
                int senha = 99;
                ContaCor cc1 = new ContaCor(titular, 0, numConta,senha);
                ContaCor cc2 = new ContaCor(titular, numConta, senha);
                System.out.println("CC1: titular: " + cc1.getTitular() + " Saldo: " + cc1.getSaldo(senha) +
" numConta: " + cc1.getNumConta() + " Senha: " + cc1.getSenha());
                System.out.println("CC2: titular: " + cc2.getTitular() + " Saldo: " + cc2.getSaldo(senha) +
" numConta: " + cc2.getNumConta() + " Senha: " + cc2.getSenha());
        }
}
```

Saída:

CC1: titular: Bob Saldo: 0.0 numConta: 30 Senha: 99
CC2: titular: Bob Saldo: 0.0 numConta: 30 Senha: 99

Item 03)
O programa não compila pois a variável d é do tipo Documento, então, mesmo que diga que a variável é referência de uma Carta ou Telegrama, não possui os métodos deles, só de Documento. Ao alterar para o main para o código abaixo, não temos mais falhas, pois há a conversão de tipos dentro dos if's, logo, o objeto possuirá os métodos então as falhas deixam de existir.
Main de ExemploPolimorfismoSemRedefinicao.java:

```
public static void main(String args[]) {
    Documento d = new Documento();
    d.imprimir();
    d = new Carta(); // d pode tambem referenciar um objeto do tipo Carta
    if(d instanceof Carta){
        Carta carta = (Carta) d;

        carta.imprimir();
        carta.anexar();
    }
    d = new Telegrama(); // d pode tambem referenciar um objeto do tipo
                                                    // Telegrama
    if(d instanceof Telegrama){
        Telegrama telegrama = (Telegrama) d;

        telegrama.imprimir();
        telegrama.registrar();
        telegrama.pagar();
    }
}
```

Item 04)
O programa não compila pois a variável c1 é do tipo ContaCor, então, mesmo que diga que a variável é referência de um ContaEsp, não possui os métodos dele, só de ContaCor, então não possui os métodos alteraLimite e getCreditoUsado utilizados. Ao adicionar as linhas para imprimirem qual classe que possui o método debitaValor, percebe-se que quando c1 = new ContaCor, é chamado o método de ContaCor, e quando c1 = new ContaEsp, é chamado o método de ContaEsp, o qual irá chamar em sua execução o método de ContaCor utilizando super.debitaValor.

Item 05)
O programa não compila ao adicionar o modificador final ao método de ContaCor, pois não permite que ContaEsp redefina o método debitaValor. Ao desfazer a adição e adicionar o modificador ao método de ContaEsp, o programa compila normalmente. O que acontece é, o modificador final determina o último ponto de redefinição, logo, se tentarmos redefinir o método além do ponto em que possui final, não conseguiremos pois ali é o máximo de redefinição permitido.

Item 06)
Não é possível redefinir o método debitaValor. Isso ocorre pelo mesmo motivo explicado no item 05, pois debitaValor de ContaEsp possui o modificador final, e ContaEspPoup tenta redefinir debitaValor.

Item 07)
Para imprimir o estado completo dos objetos Carro e Caminhão, basta chamar super.mostra() dentro dos métodos mostra() dessas classes. Ao alterar a visibilidade do método mostra() de Carro para private, quando for chamado o método mostra() de um objeto Carro no método mostraFila, será utilizado o método mostra() de Veiculo, pois é o método que está visível para o lado externo da classe Carro.

Item 08) Códigos dentro do arquivo lab05.zip.